

Technology and Implementation of
Business Rules Management Systems

Maarten Koornneef

Technology and Implementation of
Business Rules Management Systems

Maarten Koornneef

Student number: 276521

Under supervision of



drs. E. A. M. Caron
Assistant professor
Econometric Institute

dr. R. Potharst
Assistant professor
Econometric Institute



ing. J.H. Brouwer
Project leader
Working Tomorrow Amstelveen

N. van Hooidonk
Principle Consultant BPM / SOA
Financial Services Division

ing. J.P. Kooijman
Architect
Working Tomorrow Amstelveen

Acknowledgements

My gratitude goes out to all the people who have given me support and guidance during my internship at Logica. First of all I would like to mention Nico van Hooidonk, who has always made time for me and shared to me his enormous amount of theoretical and practical knowledge.

I would like to thank my supervisors on behalf of the Working Tomorrow programme, Hans Brouwer and Co Kooijman, for their feedback and project management advice. I also did not forget the guidance of Jan Adriaan Leegwater during the start-up of my internship.

I have constantly been amazed by the enthusiasm of Emiel Caron who supervised my thesis on behalf of the Erasmus University. It has been an honour to see parts of this work being used in lectures and as a subject of student assignments before I even finished writing it. I am thankful for the extra time Rob Potharst spent as second supervisor to review my thesis.

I want to express my appreciation for Huub van Thienen and François de Laender for their time and flexibility, and for Erik Noort for proofreading this thesis. My final message is addressed to Stephan Arts, Patty Brummelman, Ronald van der Plas, Mark Stevens, Rémy Vasseur and all other (ex-)students from Working Tomorrow in Amstelveen and Rotterdam: It's been a marvellous time with you!

Table of Contents

1	Introduction.....	1
1.1	Background.....	1
1.2	Research relevance.....	1
1.3	Research questions and methodology	2
1.4	Thesis outline	2
2	Knowledge Systems	4
2.1	Knowledge engineering	4
2.1.1	Knowledge acquisition	5
2.1.2	Knowledge representation.....	5
2.2	Knowledge systems	8
2.2.1	Introduction and history	9
2.2.2	Architecture.....	10
2.2.3	Advancements	10
3	Concepts and Technology of Business Rules Management Systems... 12	12
3.1	Software architectures	12
3.1.1	History and status quo.....	12
3.1.2	Service-oriented Architectures	13
3.1.3	Infrastructure of Service-oriented Architectures.....	14
3.1.4	Web services.....	16
3.2	Business rules	17
3.2.1	Definitions.....	17
3.2.2	Taxonomies and scope.....	18
3.2.3	Properties.....	19
3.2.4	Role within knowledge engineering.....	20
3.2.5	Relation with enterprise modeling.....	20
3.2.6	Relation with Business Process Management.....	21
3.3	Business Rules Management Systems.....	23
3.3.1	Architecture.....	23
3.3.2	Comparison of BRMS and expert system applications	25
3.3.3	Organizational aspects	27
3.3.4	Business Rules Management.....	28
3.3.5	Closing words	29
4	Implementing Business Rules Management Systems	30
4.1	Context of a BRMS implementation	30
4.1.1	The process of software engineering.....	31
4.2	A survey of methods and techniques.....	32
4.2.1	CommonKADS.....	33
4.2.2	STEP	36
4.2.3	RulePatterns	37
4.2.4	Rule Object	38
4.2.5	Service patterns.....	39
4.3	Analysis.....	40
4.3.1	Analyzing the survey.....	40
4.3.2	Other considerations	41
5	Case study.....	44
5.1	Background.....	44
5.1.1	Problem description.....	47

5.2	Methodology.....	48
5.3	The ILOG JRules BRMS.....	49
5.3.1	Knowledge representation in JRules	50
5.3.2	System development with JRules.....	52
5.3.3	Rule execution	53
5.4	Case study implementation	53
5.5	Discussion.....	56
6	Concluding remarks.....	58
6.1	Conclusions	58
6.2	Suggestions for further research.....	60
	Glossary.....	61
	References.....	63
A	Appendix: BKR field codes	67

1 Introduction

1.1 Background

In the field of Artificial Intelligence (AI), a rich history has been built up on Knowledge Systems (KS) and Expert Systems (ES) over the last fifty years. These systems originated as large scientific projects, then gradually matured and were introduced in commercial software applications. As the software became mainstream, academic output on the subject declined. Recently however, interest in knowledge management and knowledge system technology has made a revival in the form of Business Rules Management (BRM) and Business Rules Management Systems (BRMS).

The emergence of these systems can for a major part be attributed to the fact that the number and complexity of software systems supporting an organization, has increased over the years. This raised the need for a software architecture with clearly defined and separated tasks for every software component and standardized interfaces for communication between components. This kind of software architecture is often referred to as a Service-oriented Architecture (SOA). As a consequence, the ability of a software system to function as a component within an SOA rather than as a stand-alone software system has become a requirement.

When applied in an SOA, the design of a knowledge system needs to meet certain requirements. Additionally, a number of practical problems exist that are not dealt with by existing knowledge system implementation methods, for instance the design of SOA services. This thesis aims to increase insight in these topics by identifying the phases of a BRMS implementation process and by positioning a number of methodologies and techniques, as well as some practical issues, within this process.

1.2 Research relevance

Practitioners in the field of Business Process Management and Service-oriented Architectures can choose from a number of BRMS products today. However, each product has a different range of features and advocates a different approach to development. At the same time, the meaning and scope of some BRM concepts is not always clear and it is unknown whether existing methodologies can be applied.

As an IT services provider, Logica has interest in BRM for the following reason. Through business process outsourcing constructs, it takes responsibility for the IT infrastructure supporting the administrative processes of its clients. The company expects that a BRMS can increase the flexibility of these business processes. Therefore there exists a need for experts on BRM to apply BRMS software into the software architectures of these clients.

For academicians, this thesis tries to increase insights in how these current developments relate to traditional knowledge management concepts, methods and technologies.

1.3 Research questions and methodology

The main research question for this thesis is defined as follows: “*How can a Business Rules Management System be implemented in a Service-oriented Architecture?*”

The following sub questions are addressed in order to answer the main question:

- What is the relation between Business Rules Management Systems and Expert Systems?
- What is the relation between the concepts Business Process Management and Business Rules Management?
- Which methods can be used to implement a Business Rules Management System in a Service-oriented Architecture?
- How can these methods be combined?

Few scientific literature deals specifically with BRMS. However, much has been written about expert systems, which form the technological roots of a BRMS. In the first sub-question, the link between existing literature and this research is established, by comparing a BRMS to an expert system.

From a high-level point of view, an SOA contains an implementation of an operational business process. Therefore SOA is often associated with Business Process Management (BPM). If a BRMS is used in an SOA, an integrated approach to BPM and BRM can be taken. This is the subject of the second sub-question.

The third sub-question addresses the methodological aspects regarding the implementation process of a BRMS in an SOA. An overview of this process is presented and a number of methods and techniques are discussed that can be used during the implementation.

The goal of the fourth sub-question is to relate the discussed implementation methods and techniques to each other and to discuss to what extent they can be used in combination. Parts of the result of this are applied on a case study.

1.4 Thesis outline

The remainder of this thesis has the following outline.

Chapter 2 lays the theoretical foundation for the rest of the thesis by introducing knowledge, knowledge representation formalisms and knowledge systems. It includes a discussion of the historical developments and applications of expert systems.

In chapter 3, first the developments and trends that have led to a BRMS are reviewed, such as SOA, business rules and BRM. Next, the BRMS is described in all its facets: architecture, applications and the organizational context. The comparison with expert systems, which is the first sub-question, is made

throughout the discussion. Also the second sub-question is answered in this chapter by showing the relation between business rules and business processes.

Chapter 4 discusses and analyses five methods and techniques for knowledge engineering and BRM. One of them is CommonKADS, a well-known methodology for knowledge system development. Two methods focus explicitly on business rules. Additionally, two design- and implementation-specific techniques are addressed that focus on respectively web service and application development with a central role for business rules.

Chapter 5 presents the results of a small case study in which the ILOG JRules BRMS has been used to assess the credit history of mortgage applicants. This includes some background on a business process for mortgage offers and a technical discussion of JRules.

In chapter 6 conclusions and suggestions for further research are discussed.

2 Knowledge Systems

Knowledge is power — goes the saying. It is not without reason that in the current information-oriented society, knowledge is regarded as the third economic resource, after labor and capital. Yet, it is an abstract and intangible asset. The meaning of knowledge can be clarified by relating it to the concepts *data* and *information*.

According to Schreiber et al.,

Data “*are the uninterpreted signals that reach our senses every minute*”, without a meaning attached to it. The ones and zeros on a computer hard disk, and lines and colors from a drawing are examples of “raw” data.

Information “*is data equipped with a meaning*”. The ones and zeros form a picture file or a document, for example. Lines and colors are part of a graph containing sales figures.

Knowledge is information applied in a specific *context* or *domain*, with an attached *purpose* or *action*. For example, decreasing sales indicate that a certain response is needed. Knowledge has also the capability to generate new information, namely through reasoning [S+99].

Humans possess knowledge, but also a computer system can store knowledge and reason with it. Such a system is called a *knowledge system*. The process of building a knowledge system is called *knowledge engineering* [Dur94].

The remainder of this chapter describes various aspects of knowledge engineering and knowledge systems in more detail.

2.1 Knowledge engineering

Various sources, among which [Dur94] and [S+99], mention how the knowledge engineering process can be broken down in a number of phases. Mostly there are not many differences between the proposed breakdowns, although some sources give different names to certain phases, have one or two extra phases or lay certain emphases. In general, the phases of knowledge engineering mentioned below reasonably overlaps most commonly used breakdowns:

1. *Strategy definition*. In this phase, the strategy and goals are formulated, problems and potential solutions are identified and the feasibility of these solutions is determined.
2. *Knowledge acquisition*. In order for a knowledge system to use knowledge, it must first be acquired. This activity involves collecting knowledge from various sources.
3. *Modeling and design*. Past experiences have learned that knowledge cannot be directly “transferred” into a knowledge system [S+99]. Therefore, it must first be modeled and represented into a formal *representation format* that a knowledge system can understand.

4. *Implementation*. This involves entering the knowledge in a system and creating documentation of the knowledge and the system's reasoning methods.
5. *Testing*. Implementing and testing are often done concurrently. The goal of testing is to validate the reasoning of the system.
6. *Maintenance*. After the system has been put into production, changes to the system may be desired. If this is the case, the process will be started again from activity 1.

It is beyond the scope of this thesis to elaborate on all phases. However, *knowledge acquisition* and *knowledge modeling* are considered important. Therefore, these subjects are explained below.

2.1.1 Knowledge acquisition

Before a system can use knowledge, it must first be acquired from a certain source. The two major types of knowledge sources are documented (written sources) and undocumented (humans). Sources of documented knowledge include reports, books, guidelines, legal documents and so on. Documented knowledge is also called *explicit*, whereas undocumented knowledge is called *tacit* knowledge [Pol58].

It can be difficult to capture knowledge, and it is even regarded as the bottleneck in knowledge system development [Byr95]. Also some kinds of knowledge are harder to capture than others. This can be especially hard if the knowledge is for example ambiguous, cannot easily be stated in a formal representation, or the knowledge has to be separated from lots of other information that is considered unimportant.

Finally, knowledge can be acquired manually or automatically, or using a combination of both. Manual methods include interviewing people with "expert" knowledge and analyzing documents. The term *data mining* is often used for forms of knowledge acquisition that are (partly) automatic.

2.1.2 Knowledge representation

For humans it is completely obvious to use natural language to understand and communicate knowledge, and to reason with it. Despite the progress that is made in the field of Natural Language Processing (NLP), it turns out that for a computer it is much more difficult to do, because natural language is inherently ambiguous. Therefore, a computer needs a formal and unambiguous format in order to use knowledge.

In this subsection, a number of well-known *knowledge representation formats* will be mentioned briefly that knowledge systems use to work with different kinds of knowledge. A more detailed discussion can be found in almost all textbooks on knowledge systems. In this thesis, particularly the following properties of knowledge representation formats are discussed:

1. *Expressiveness*, or: how easy it is to express something in a particular format. Depending on the format, it may or may not be straightforward

(or even possible) to make certain kinds of statements. Other formats may be bound to specific domains.

2. *Interpretability*, or: how easy it is to understand the semantics of a particular format. In many cases it is important that knowledge is represented in such a way that both a human and a computer can understand it. However, knowledge constructs that are easily interpretable for a human, are often hard to understand for a computer and vice versa.
3. *Inference mechanisms*, or reasoning methods. The ways to draw conclusions or infer new knowledge from existing knowledge vary per representation format. Also humans and computers do this in different ways.

Often a knowledge system uses one or more of the following knowledge representation formats: objects, formal logic, production rules and structured language.

Objects

Although *objects* are a relatively new form of knowledge representation, they have been adopted rapidly as a well-known form of modeling the static structure and behavior of entities. They are also used in Object-oriented Programming (OOP), currently a standard programming paradigm.

It is possible to look at objects in different ways. Also the way how objects are implemented in programming languages and other systems varies slightly. From a conceptual view, an object is described by its name, properties, methods and parent object. The properties and methods describe respectively the state and the behavior of an object. An object may also inherit properties and methods from another object, which is called its *parent*. These properties have a certain type (i.e., data type, range etc.) and can be accessed and mutated through *get* and *set* functions. Finally, an object can be instantiated. An *instance* of an object represents an actual example of the object, rather than its description.

Although objects became widespread only in the nineties of the previous century, they are closely related to older forms of knowledge representation, particularly *frames* [Min75] and in a lesser extent *semantic networks* [Sow87] and *entity-relationship* diagrams.

Frames have been used extensively in knowledge systems. Frames have *slots* (properties), can be organized hierarchically and can be instantiated. Also a frame may include *facets* such as IF-NEEDED and IF-CHANGED to respectively access and mutate slot values. Facets can have *procedural attachments* (*daemons*), which specify actions that can be executed, which resemble the get and set functions of objects.

Semantic networks and entity-relationship (ER) diagrams are graphical modeling methods. In a semantic network, various terms are notated in a balloon. The relation between two terms is indicated by an arrow between balloons and a description of the kind of relation. Entity-relationship (ER) diagrams are mostly used to describe data models.

Objects and frames have the advantage that they are easily understandable for humans, while a machine can also interpret them. However, the expressiveness and capabilities for reasoning are limited for both forms when they are used as the only form of knowledge representation. Therefore, frames and objects are mostly used in combination with other forms of knowledge representation rather than alone [Tur05].

Formal logic

Formal logic forms such as *predicate logic* are a mathematical way of describing assertions, attributes and relations. In predicate logic, the rule “All clients having a mortgage, must own a house” would be expressed as follows:

$$\forall x: \forall y: \text{Person}(x) \wedge \text{Mortgage}(y) \wedge \text{IsOwnerOf}(x,y) \rightarrow \text{OwnsHouse}(x)$$

The syntax of predicate logic and related forms of knowledge representation will not be discussed here, but a detailed introduction can be consulted in [LG91]. Using predicate symbols, a *fact base* can be constructed.

This representation format has a number of nice mathematical properties, for instance: domain knowledge can be expressed in a purely declarative logic format and can be separated from inference rules (procedural logic) [RN03]. These mathematical properties make this suitable for a knowledge system to reason with. On the other hand, anyone who is not a mathematician will have trouble understanding this format. Also, according to Lucas and Van der Gaag, its expressiveness is limited to only certain kinds of knowledge, and “*When in a specific problem domain the knowledge is not available in a form ‘close’ to logic, a lot of energy has to be invested into converting expert knowledge to logical formulas, and in this process valuable information is often lost*” [LG91].

Production rules

Production rules have the basic form “IF <condition(s) are fulfilled> THEN <draw conclusion(s)>”. The form “WHEN <condition(s)> DO <conclusion(s)>”, which is semantically the same, is also sometimes used. Production rules have been used successfully since the seventies in the past century in so-called *rule-based systems* while the underlying model of reasoning with *production systems* is even older. The conditions in a production rule are always tested against a domain knowledge model. This domain knowledge can have the form of object-attribute-value (o-a-v) triplets or can use an object-oriented or frame-based model. With the evaluation of the conditions, a technique known as *pattern matching* is often used. In the conclusion of a production rule, new facts or actions can be specified. Production rules can represent various kinds of logic, such as causal relationships, heuristics and directives. [Dur94] provides a practical guide on production rules and rule-based systems, while [LG91] is a comprehensive source on building rule-based systems.

Compared to other representation formats, production rules provide a middle course with respect to interpretability because they are easier to understand for humans than mathematical logic, while a computer can still interpret them and reason with them. However, when a knowledge base contains many production rules, maintenance can be problematic [LG91].

Reasoning with production rules is possible through *forward-* and *backward chaining*. In the simplest possible case, this works as follows. Consider a set of rules containing two rules:

- If a customer has a *Gold* ranking, then he applies for a 10% discount.
- If a customer has no late payments, then he has *Gold* ranking.

Now an information system has the task to determine whether a person with customer number #004237 applies for a 10% discount, while its working memory already contains that fact that that this customer has no late payments. A forward chaining system iteratively finds all rules of which the conditions are true, and executes the conclusions. In the first iteration, rule 1 does not *fire*, because the ranking is unknown at this moment, while rule 2 fires. The fact that this customer has *Gold* ranking is then stored in the working memory. In the next iteration, rule 1 will fire, the conclusion is drawn that the discount will be given to the customer. A backward chaining system works the other way around. It starts with the goal and then tries to solve all variables that are needed to reach a conclusion. Such a system executes rule 1 first and concludes that it needs to know the ranking of the customer. By executing rule 2 it obtains this fact.

Related to this form of knowledge representation are *decision tables* and *decision trees*, which can be translated into production rules without losing information, so technically, these can be implemented as (a special kind of) production rules.

Structured language

Structured language is a subset of natural language that conforms to a predefined vocabulary or template to make it less ambiguous and to limit its usage to certain constructs or domains.

Because of the removed ambiguity, structured language can be easier for a human to understand than natural (unstructured) language. By contrast, for a computer this format is still relatively hard to understand, although successful attempts have been made to make a mapping from structured language to some formal way of expressing logic that a computer can execute or reason with. A number of knowledge systems exist in which knowledge is presented to the human user as structured language, but internally represented by interpretable language or some formalism [Gra06].

2.2 Knowledge systems

Some authors use the terms *knowledge-based system*, *knowledge system* and *expert system* interchangeably. For this thesis however, the first two terms are used as synonyms and have the meaning as described by Schreiber et al. [S⁺99]: they can be distinguished from “normal” software systems by the distinctive property of having “*some explicit representation of the knowledge [that] is included in the system*”.

According to Durkin, an expert system is “*a computer program designed to model the problem-solving ability of a human expert*” [Dur94]. Lucas and Van der Gaag have defined it as “*a system ... capable of offering solutions to specific problems in a given domain or able to give advice, both in a way and at a level comparable to that of experts in the field*”

[LG91]. Most other definitions do not differ much from the previous two. In the rest of this thesis the term *expert system* is used in a way that overlaps most common definitions:

An Expert System is a knowledge system that emulates a human expert.

Although maybe any system that uses AI techniques might to some extent be regarded as an expert system, this thesis focuses on “classic” expert systems and particularly *rule-based* expert systems, which use (production) rules as the main representation and reasoning mechanism. The remainder of this chapter contains a discussion on the first generation of these expert systems, which were built in the 1970s, and the developments that have taken place since then. These developments consisted essentially of adding new features and components to the architecture of this first generation.

The discussion that follows is not meant to give an comprehensive introduction on expert systems and their applications. Instead, some characteristics are highlighted and commented on. Readers interested in a more complete description are encouraged to read [Dur94] first to get an overview.

2.2.1 Introduction and history

Early expert systems were built with the purpose of solving decision-making problems that required a lot of expertise. During system development this expertise was gathered from human and written sources by for instance interviewing one or more experts or analyzing documents or data, a process known as *knowledge acquisition*. Then it was translated into a machine-interpretable format, often production rules. In order to use the system, the knowledge user could connect to it via a terminal. This system then started a dialogue with the user in a way that resembled the way an expert would operate. In this dialogue, the system asked questions to the user. When the user had typed an answer, the system used forward or backward reasoning (or both) and determined what further information was needed from the user. Then subsequent questions were asked until a conclusion could be drawn. It was also considered important that the user could ask the system why it asked a specific question, or how it had reasoned to draw certain conclusions, just like human experts would explain their reasoning if asked to do so. Therefore, when a user typed the phrase “HOW” during a conversation, the system would return a rule trace to show how it had reasoned. By asking “WHY”, the system returned either what it would conclude when a certain answer was given, or the rule it currently pursued.

In that age, the majority of people were not accustomed to working with a computer. Nevertheless, the end users had to trust the system and treat its answers as they would treat a human expert. In that time, a lot of effort had to be made to overcome their skepticism. Davis et al. reported on this issue: *“The systems’ acceptance will be strongly dependent upon the extent to which its performance is natural (i.e. human-like) and transparent. Lack of acceptance of some application programs can be traced to their obscure reasoning mechanisms which leave the user forced to accept or reject advice without a chance to discover its basis.”* [D+77].

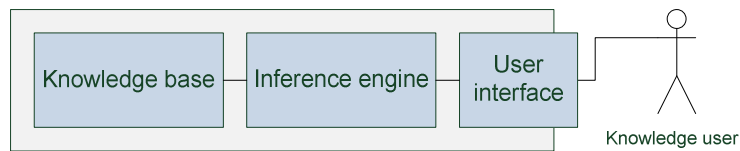


Figure 1. Architecture of a traditional expert system.

2.2.2 Architecture

A reference architecture for a traditional expert system could look like Figure 1. Most sources mention only the *knowledge base* and the *inference engine* as the two main components, but the choice to include also the *user interface* here was made deliberately in order to make a comparison to knowledge systems of later origin in chapter 3. The individual components will be discussed below.

The knowledge base, also called *rule base* in case of a rule-based expert system, contains the domain knowledge of the system and serves as its *long-term memory*. The fact that it is a separate component does not mean that there is an editor for rules integrated in the system – at least not in the early generation of expert systems. These systems required programming in a language such as OPS, LISP or PROLOG to change the rules in the knowledge base.

The inference engine uses AI techniques to decide on the problem. This component includes a *working memory* or *short-term memory* containing the facts that are established so far from given answers by the user and through inference. This short-term memory also contains an agenda with the next rules to be pursued. Finally, the explanation facility that answer the HOW and WHY questions can also be regarded as a part of the inference engine, although it is also sometimes seen as a separate component.

The user interface outputs questions to the user and reads and interprets given answers. A spelling checker can also be present. In case the user makes a spelling error, the system matches the given input to the list of valid answers and suggests a correction.

2.2.3 Advancements

During the 1980s, the number of applied expert systems grew exponentially and they started to be applied to solve various types of problems such as diagnosing, interpretation of data, planning, prediction and recommendation. Application areas included agriculture, manufacturing, finance and accounting. Also multiple kinds of knowledge representation and “intelligent” techniques started to be employed such as case-based reasoning and various search techniques.

Two of the most important developments in expert systems have been the expert system builder and the hybrid expert system. When these two are combined, a system is formed with an architecture as shown in Figure 2.

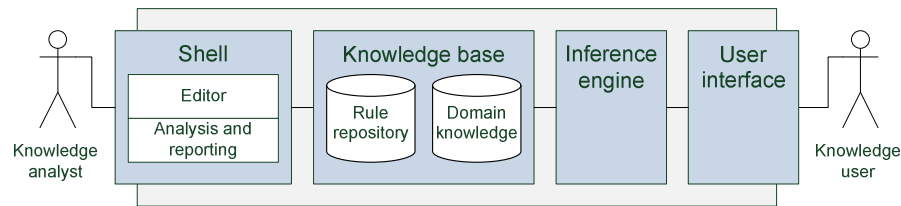


Figure 2. Architecture of a hybrid expert system builder.

An *expert system builder* is a software system in which a development environment called a *shell*, is added to the expert system. By removing all knowledge from the knowledge base, an “empty” expert system is made, which can then be used for any problem area by filling the knowledge base with knowledge appropriate for a particular problem area. Therefore, an expert system builder is *domain-independent*. By using an editor, understanding of a programming language is no longer required to fill the knowledge base. Furthermore, the shell may include analysis and reporting functionality that, for instance, is able to detect inconsistencies in the knowledge base, knowledge that is never used or other features that can be used during maintenance and testing. Therefore, the knowledge engineer no longer has a role as a programmer, but rather as a *knowledge analyst*.

The second important development was that *hybrid expert systems* appeared in which multiple forms of knowledge representation were employed. In such a case, often the terms and facts that are used in the rules, are specified in a *domain knowledge model*. Such a model can be implemented as a semantic network, an entity-relationship (ER) model or an frame-based or object-oriented model. Then the inference engine must match the objects with the rules. A number of algorithms exist that do this efficiently, most of which descend from the Rete pattern matching algorithm [For82].

3 Concepts and Technology of Business Rules Management Systems

Business Rules Management Systems can be regarded as a new generation of knowledge systems, which are meant to be used as decision-making systems within large-scale software architectures. The criteria for these decisions are stored inside a repository for *business rules*. The rise of BRMS can for a major part be attributed to developments in software architectures on the one hand, and advancements in knowledge system development facilities on the other hand. At the same time it can be observed that to a growing extent software architectures contain an explicit model of the business process they support. Essentially, the role of a BRMS in such an architecture is to facilitate an explicit model of the knowledge used in particular activities of the business process.

This section first discusses software architectures and business rules separately. Then the architecture of a BRMS is explained, followed by its application domains and the impact of a BRMS on roles in an organization.

3.1 Software architectures

3.1.1 History and status quo

Many organizations currently use a variety of software systems to support and automate business tasks. These include software systems for inventory, accounting, invoicing, customer relationship, payroll, document management, and so on. In the past decades, the amount and complexity of these software systems has steadily increased, and it is likely that this trend will continue in the foreseeable future. Furthermore, these systems do not function stand-alone anymore, but interaction between systems is required in order to exchange data or messages or to distribute functionality. The complexity of the total of information systems is particularly a problem in large organizations that have gone through multiple phases of automation throughout the years and in the end use a combination of new and old (so-called *legacy*) systems. Also mergers and acquisitions (M&A) contribute to this complexity because systems from what have been different organizations, need to be integrated. Because of this, it has become increasingly important that individual systems and communication between them are clearly defined.

Also individual systems have increased in complexity, which has raised the need for a clear definition and separation of functionality of software components. Around the year 2000, Van der Aalst described some highlights in the evolution of information systems architecture [AH00] and distinguished the following 4 periods.

1. 1965-1975: information systems are stand-alone and often monolithic applications in which data or message exchange between systems is impossible in electronic form. All applications have their own components for data management, user interfaces and so on.
2. 1975-1985: data organization, storage and retrieval are externalized from the applications to database management systems (DBMS) which are permanently available and could be accessed by any software system.
3. 1985-1995: user interfaces standardize and the generic parts in them are externalized to user interface management systems (UIMS) that can be part of the operating system.
4. 1995-2005: now that software designers do not have to worry about the implementation aspects of data and user interface management anymore, the main task of applications is to handle cases and business processes. These business processes can be externalized as well, and become part of workflow management systems (WFMS) or business process management systems (BPMS). This can reduce the time needed to build and maintain applications, and the time needed to implement a change of the business process in information systems.

3.1.2 Service-oriented Architectures

The paradigm of service-orientation is more and more being adopted with the intention to manage the complexity of the information system landscape in an organization. When service-orientation is applied both on the level of individual applications and the aggregate application infrastructure, an organization is said to have implemented a Service-oriented Architecture (SOA). Within an SOA, the concept of a *service* plays a central role. A service can be seen as an independent (software) component that performs a clearly defined task and can interoperate with other services.

In the recent past, a lot has been said and written about SOA and many promises have been made by information system practitioners and vendors of software integration tools regarding problems that SOA would solve. A few years ago this occurred in such an extent that one could legitimately speak of a hype. At the same time, a cloud of mystery surrounded the term because many different definitions and interpretations of “service” and “SOA” were being used.

Fortunately people such as Erl [Erl05] have contributed to demystifying all this. In the rest of the discussion on SOA in this section, much has been based on his book. He initially presents service-orientation as a design philosophy, which can be applied to information systems architecture. Conceptually, this has nothing to do with any implementation technology or platform. However, certain technologies and infrastructure models exist that have proven to be particularly suitable to implement the principles of an SOA, and in practice these are almost always used.

Fundamentally, SOA is according to Erl:

“an ideal vision of a world in which resources are cleanly partitioned and consistently represented” [Erl05].

He continues:

“By adhering to this vision, past technical and philosophic disparities are blanketed by layers of abstraction that introduce a globally accepted standard for representing logic and information.”

OASIS (Organization for the Advancement of Structured Information Standards), a leading authority on the area of SOA technologies and standards, uses a different definition and calls it:

“A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations” [Oas06].

Although both definitions are widely accepted, for the purpose of this thesis they are too general. Therefore a different definition is used here which is based on the ideas of Erl and loosely on the OASIS definition:

A Service-oriented Architecture is a paradigm for the design of an IT-infrastructure that consists of autonomous and loosely coupled components that individually perform a clearly defined and unique task and work together seamlessly and consistent with business objectives.

In an SOA, information systems functionality is cleanly partitioned into services. Communication of information between services takes place using standardized protocols. This information is structured and represented in a standardized way.

In order to implement an SOA, the technology platform of XML Web Services is almost always used (simply called *web services* hereafter). This platform makes use of a extensive number of protocols and standards to provide various communication facilities such as describing services, messaging, coordinating between services, handling transactions and security, describing business processes and so on. There are three organizations that establish these standards: OASIS, the W3C and WS-I. Although web services are not the first platform to provide this functionality, it offers possibilities for communication of information on a level that lies beyond what has been possible with earlier technologies such as CORBA, due to for example the amount of standards that is available.

3.1.3 Infrastructure of Service-oriented Architectures

An SOA is often associated with a layered architecture, which is the way how organizations most often implement it. This architecture is being characterized by the fact that each layer contains services that communicate only with those other services that reside in the layer directly above or underneath them. Such a model is presented in Figure 3. Different pictures exist of this model with slight variations, but they are all based on the same ideas. Broadly speaking, the services in each layer perform a task at a lower level than the services in the layer above it.

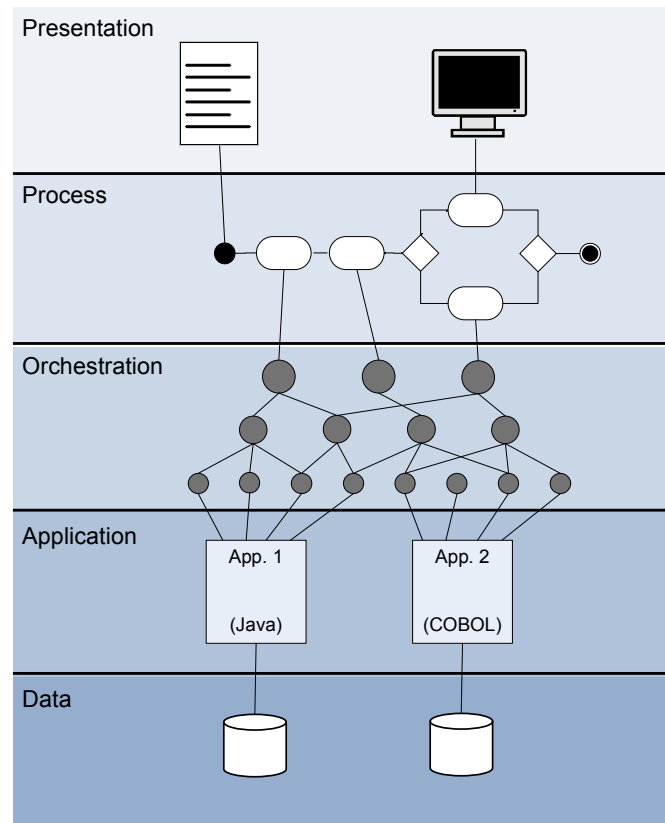


Figure 3. Typical infrastructure of a service-oriented architecture

The *presentation layer* is the link to the outside world. This can be a human end user from inside the organization working via an intranet web application, or a customer who is making transactions over the Internet from a website, but also an Electronic Data Interchange (EDI) interface, an e-mail or SMS gateway, a fax machine and so on. Events in this layer often trigger activities in the business process. For instance, a web site visitor clicks on the Submit button to book a flight, or an office employee processes a case through a workflow system.

The *process layer* contains services that have a clear correspondence with activities or tasks in the business process. The services in this layer may be contained in a BPMS.

The *orchestration layer* acts as an intermediate between the business activities as defined in the first layer and technical implementation of these activities in the third layer. This layer in turn can be composed of multiple sub-layers. Each of these can pass on (“orchestrate”) tasks to one or more other services in the same sub-layer or the one underneath it, such that approximately a tree structure is formed of services that are coupled to each other.

The applications in the *application layer* need to expose their functionality through web services (or another SOA-compliant interface). Older applications cannot do this and must first be modified to provide such an interface. Alternatively, an

Enterprise Service Bus (ESB) may be placed between these applications and the web services as an extra layer to convert messages from a web service-compliant format to the built-in format used by these application. An ESB can for example be used to connect a web services environment to a mainframe environment that uses CICS transactions.

Finally, the *data* layer contains the databases used by the applications in the above layer.

In an organization, changes may occur on technical implementation level, such as an application upgrade to a newer version or a merge of multiple databases as a result of M&As. This does however not change the business process, so these changes do not affect the process layer. In other situations where changes occur in the business process, in some cases this leaves the application layer unchanged (except when functionality is required that was not implemented before the changes). Therefore, the externalization of business processes from information systems as described in section 3.1.1 has been realized with this infrastructure, as well as the externalization of data and user interfaces.

3.1.4 Web services

A large amount of standards is available for SOA that uses web services. These standards provide ways to communicate information between systems and use text files in the XML (Extensible Markup Language) format.

One of the most important standards is WSDL (Web Service Description Language), which describes the inputs, outputs and communication pattern of a web service, as well as the used data types for the inputs and outputs and some other information. The most commonly used communication patterns are:

- *Request-response*. This type of web service can receive a request from another web service and will send a response.
- *One-way*. This type of web service can receive a request and does not send a response.

Hence, conceptually the way how a request-response type of web service is invoked resembles the way how a method in a conventional programming works: it is invoked by another method, has a number of parameters and returns a value. By comparison, the one-way type of web service can be compared to a method that does not return a value.

The mostly used message format for the request and response messages is SOAP (Simple Object Access Protocol). Examples of WSDL and SOAP can be found in any reference material about web services.

3.2 Business rules

In recent years, the term *business rule* has gained an increasing amount of popularity in some communities, particularly certain information technology practitioners and BRMS vendors. Examples of business rules include:

- *“A customer will get a discount of 10% on his purchase, if he buys more than three items of product X”*

and

- *“If an offer has been sent to a customer and he has not responded after three days, a sales representative will call him back to ask whether he is still interested”.*

Although it is fairly easy to get an intuitive understanding of the meaning of the word “business rule”, at closer inspection it turns out that the various definitions that exist, do not completely overlap in terms of meaning and scope. Also some sources define the concept improperly or not at all, which can lead to misunderstandings. Some other authors mix up their definitions with what they regard as best practices.

In this thesis a brief overview is presented of various definitions and interpretations of business rules. This is followed by a classification scheme and properties of business rules.

3.2.1 Definitions

In the literature about business rules many different definitions are found. Some authors give definitions spanning multiple sentences of more than 40 words in total [Gra06], while others use the term and give examples, but do not give a clear definition at all [Dat00, Deb05]. The Business Rules Group (BRG), an organization of system and business analysis methodologists, has defined a business rule as *“a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business.”* [Bus00]. Other definitions include: *“rule under business jurisdiction”* (where “rule” has the meaning of *“guide for conduct or action”*) [Bus05], *“the means by which an organization implements competitive strategy, promotes policy, and complies with legal obligations”* [HG06], *“a statement about how the business is done, i.e., about guidelines and restrictions with respect to states and processes in an organization”* [Her95, B+90], *“a means of describing an organization’s policies”* [PPL98] and *“A compact statement about an aspect of a business”* [Mor02]. The author of the last definition adds that *“[a] rule can be expressed in terms that can be directly related to the business, using simple, unambiguous language that’s accessible to all interested parties: business owner, business analyst, technical architect, and so on”*. According to [Gra06], most older sources about business rules use the term to describe certain kinds of database constraints.

In an effort to clarify the concept (thereby hopefully *not* adding to the fuss unintentionally), this thesis uses the following definition:

Business rules are statements that express the criteria for operational business decisions.

Structural assertions	
Term	The head office
Fact	The head office is located in Amsterdam
Action assertions	
Constraint	A customer may order between 1 and 500 items
Action trigger	If the stock level is below 50, then replenish
Derivations	
Calculation	Profit equals revenue minus costs
Inference	If the stock level is below 10, then the state is "critical"

Figure 4. Examples of business rules according to the BRG classification.

Business rules hence describe the course of action deliberately taken by a business. The word *rule* should be interpreted here in an informal sense as “rule of conduct” or “rule of the game”, rather than in a formal sense as “production rule”. Business rules reflect the way how a business implements its competitive strategy and complies to legislation [HG06]. These rules are used and enforced on the operational level, whereas their definition and design takes place on the tactical level.

3.2.2 Taxonomies and scope

A classification scheme increases insight in what an author considers as the scope of the concept “business rule”. The Business Rules Group distinguishes the following types of business rules:

1. *Structural assertions*, which are either “terms” or “facts”, in the usual meaning of the words.
2. *Action assertions*, which test a certain condition. This kind can be split up further in a number of ways. One way is to distinguish conditions that must always be true (“integrity constraints”) to keep a valid state, conditions that enable an action or trigger an event when true, and authorization rules for certain actions. Another way to categorize action assertions is by distinguishing hard (strict) and soft (guideline) action assertions.
3. *Derivations*, which are either mathematical calculations or inferences. In the latter, new facts are derived from known ones using induction or deduction.

Figure 4 shows examples of these types. Most other authors who use their own classification schemes or otherwise disagree with the BRG, use a narrower scope of the term. For example, Graham excludes structural assertions and calculations [Gra06]. Others, for example Chisholm and Date, take a point of view on business rules that is related to relational database design and emphasize (complex forms of) integrity constraints in particular [Chi04, Dat00].

A closer look the taxonomy of the BRG reveals that business rules are closely related to the more general term “knowledge”, as all three mentioned types of business rules can be expressed with certain knowledge representation formats:

structural assertions can be represented with objects, action assertions can be represented with production rules and (integrity) constraints, and derivations can be represented by formal logic or production rules. In later sections the relation between business rules and knowledge is characterized.

3.2.3 Properties

Various sources mention a number of properties of business rules. As mentioned before, a problem is that some authors do not make clear whether they regard certain aspects as inherent properties of business rules or rather guidelines for use. Therefore we first take a neutral point of view and describe only the properties that are most often encountered. Then an attempt is made to express the relation between business rules and knowledge.

Firstly, several authors mention *atomicity* as a property [Bus00, Gra06]. In the case of business rules, atomicity means that they cannot be broken down further without losing information, so business rules are the most low-level form of expressing decisions. On the other side of the spectrum, higher-level descriptions of decisions include strategy definitions and business policies. The BRG has formalized this spectrum in a conceptual model in which they describe terms related to business rules in an entity-relationship diagram. In this model, they define the concepts *policy*, *business rule statement*, *formal rule statement* and more, and how all of them are related to each other. According to their point of view, a policy is “*a general statement of direction for an enterprise*” and may be composed of more detailed policies. A policy may be the basis for one or more business rule statements, which in turn can be broken down into one or more (atomic) business rules. The last can optionally be represented by a “formal rule statement”, which uses a structured language form to represent the business rule.

The second property which almost all sources on business rules mention, is that they (should) have no implicit or explicit ordering [Bus00, Dat00, Gra06, Mor02]. Hence, they can be regarded as *declarative* knowledge. With declarative knowledge, the control information that is needed to use the knowledge does not reside in the knowledge itself. By contrast, in *procedural* knowledge, this control information is embedded in the knowledge [RN91]. Information about the order in which a set of statements in a knowledge base should be interpreted, is an example of control information. This means that any possible or desirable states are only *described* and nothing is said about what *steps*, *procedures* or *transitions* must be taken to reach these states [BRG00]. Date explains this by stating that business rules describe “*what needs to be done instead of how*” to do that [Dat00]. This also means that process logic are not business rules. Furthermore, it implies that the ordering of rules (which is procedural knowledge), lies outside the scope of the concept “business rule”. As a side note, this also indicates that most conventional programming languages are badly suited to represent business rules, because a source code listing contains instructions that are executed from top to bottom. Source code therefore is a form of procedural knowledge. By contrast, production rules *can* be declarative knowledge, but this depends on the way they are interpreted.

The representation format of business rules could also be regarded as a property. Some authors use the term “business rule” to refer to statements that are “*well-formed*” with respect to (i.e. convertible to) some kind of formal representation

format [Dat00, Gra06], while for example Morgan states that business rules are expressed in “*simple and unambiguous language*” [Mor02]; something that cannot be said from some formal knowledge representation formats. Therefore it seems that there is no consensus on whether business rules have by definition a certain representation format or not.

3.2.4 Role within knowledge engineering

Business rules are related to knowledge modeling and knowledge representation. Following the earlier mentioned literature sources on business rules, we believe that the role of business rules within knowledge engineering is as follows:

As mentioned in section 2.1, one of the phases in knowledge engineering is modeling. Schreiber et al. advocate a top-down approach to knowledge modeling: knowledge is first described at a high level in terms like business objectives and activities in a business process. This high-level knowledge is then broken down in a number of steps, until a detailed knowledge model is constructed. This knowledge model is defined in an implementation-agnostic way. During the implementation phase, this knowledge model is translated to a technical implementation in the form of programming language constructs or knowledge representation artifacts.

Seen within the process of knowledge engineering, the definition of business rules can be regarded as the final step in knowledge modeling. Therefore, business rules are the most low-level blocks of knowledge, they have a business context and they are represented in an implementation-agnostic way.

3.2.5 Relation with enterprise modeling

Enterprise modeling and enterprise architecture deal with describing aspects of an enterprise. These aspects include data, descriptions, workflows, etcetera. Each aspect needs to be modeled in a different way. Business processes for example are described in a business process model, and employee roles are described in organization charts. Business rules are also an aspect of an enterprise, because they describe criteria for operational decisions.

The Zachman Framework [Zac87, SZ92] is a framework for enterprise architecture. It presents an enterprise architecture meta-model in a 5 by 6 matrix scheme where each column describes a different aspect of the enterprise, and each row represents different perspectives from which the aspects can be seen. The names of the rows and columns with examples of cell contents are shown in Figure 5. The ordering of the columns is arbitrary, but every row describes the business at a lower level than the row above it. Every cell is unique in the sense that it describes a distinct aspect seen from a distinct viewpoint. Different graphical representations and models are appropriate for different cells. For example: activity diagrams describing business processes would belong in the *function* column and the *owner* row. In the *motivation* column in the *planner* row, the list of business goals should be placed. Below row 5, at the lowest level, the actual instances of the business entities can be placed, such as the actual data, people and so on. However, by incorporating actual instances, we have gone outside the boundaries of what is called “architecture”.

	Data (What)	Function (How)	Network (Where)	People (Who)	Time (When)	Motivation (Why)
Scope (Planner)	List of things	List of processes	List of locations	List of organizations	List of events	List of goals
Business model (owner)	Semantic model	Business process model	Business logistics	Workflow model	Master schedule	Business plan
System model (designer)	Logical data model	Application architecture	Distributed system architecture	Human interface architecture	Processing structure	Business rule model
Technology model (builder)	Physical data model	System design	Technology architecture	Presentation architecture	Control structure	Rule design
Detailed representations (sub-contractor)	Data definition	Program	Network architecture	Security architecture	Timing definition	Rule specification

Figure 5. The Zachman Framework for Enterprise Architecture matrix.

The logic behind the model was inspired by the way how buildings and airplanes are designed and constructed. According to [SZ92], the model could be applied to the design of anything that is complex and where the different perspectives make sense.

At the time the model was created, not for every cell the exact meaning was clear. This held for example for some viewpoints in the *people*, *time* and *motivation* columns, for which it was unknown how they should be used in practice. Also the relations between various cells are not always trivial. In subsequent publications, the descriptions have changed for some cells as a result of progressive insights. Later on, it has been recognized by for example the BRG that business rules belong in the *motivation* column of the Zachman Framework, because the policies that lie on the basis of the rules, are the implementation of the business strategy, which is actually the *motivation* seen from the *planner* viewpoint. The detailed rule specification would reside in the *sub-contractor* row, because a rule specification gives a model of a business policy at most detailed level.

3.2.6 Relation with Business Process Management

According to Van der Aalst, “*Business Process Management includes methods, techniques, and tools to support the design, enactment, management and analysis of operational business processes*” [AHW03]. A business process consists of a collection of activities that are executed in a predefined order and contribute to a clear business goal. In most companies, production processes are the primary processes, while for example marketing, stocking, sales and invoicing are considered secondary. To each individual activity, a number of business rules may be applicable. While some business rules can be attributed to one specific activity in a process, other examples are thinkable where business rules are “shared” between multiple activities or processes.

Some activities can be classified as knowledge-intensive. Examples of knowledge-intensive activities include classification, assessment, planning and scheduling [S+99]. In these activities a (potentially) complex decision is made, based on a number of business rules.

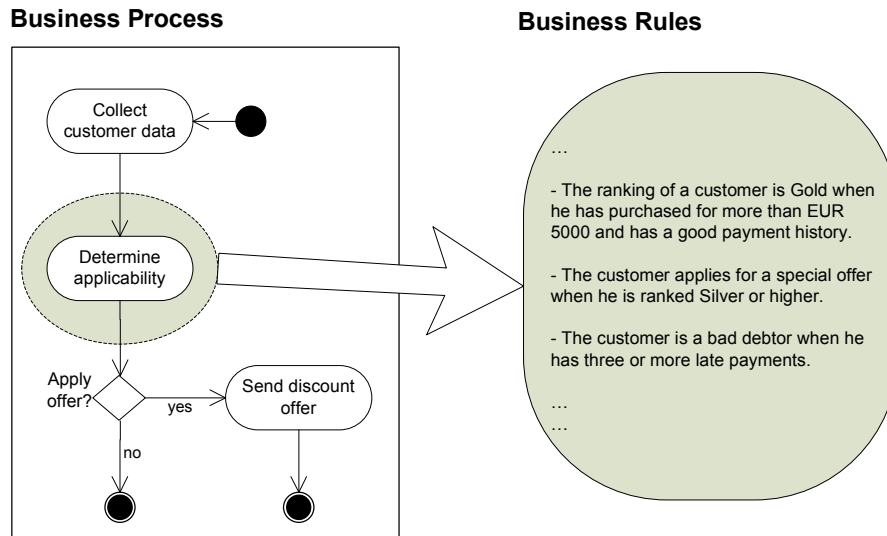


Figure 6. The relation between business processes and business rules explained by a fictional business process.

The relation between business processes and business rules can be explained by a fictional example case in which a company wants to send a “special discount offer” to a selected subset of its customer base. The selection criteria include the likelihood that the customer will accept the offer and the likelihood that the customer will pay on time. A schematic display of the business process may look like the UML activity diagram on the left pane in Figure 6. At the start of the process, data is collected from a certain customer. Because the company needs data to evaluate its selection criteria, it collects the client’s payment history, loyalty indicators and some measure of the likelihood he will accept the offer. In the next activity, it is determined whether the offer will be sent to a particular client or not. The activity “determine applicability” is an example of a data- and knowledge-intensive task, which is carried out by evaluating a number of business rules against customer data. These rules may be a mixture of specific rules for this activity and general rules.

The outcome of the decision is not directly present in these rules, but is obtained through reasoning. Let’s say that we need to decide on a customer who has purchased for EUR 6500 so far and it has already been concluded that he has a good payment history. Now we can combine the first and the second rule to conclude that this customer has a *Gold* ranking, which is higher than *Silver*, so he applies for the special offer. Note that this rule list is incomplete, because the criteria for *Silver* and (maybe) other rankings is undefined. So if a customer is a *bad debtor*, it still cannot be told for sure whether he applies for a discount or not, because it is unknown which ranking he has. Of course, the rule list can be completed by adding the criteria for other rankings, more detailed descriptions of what is considered a good payment history and so on.

When the applicability of the customer has been assessed, its further course in the business process is determined. The relation between business processes and

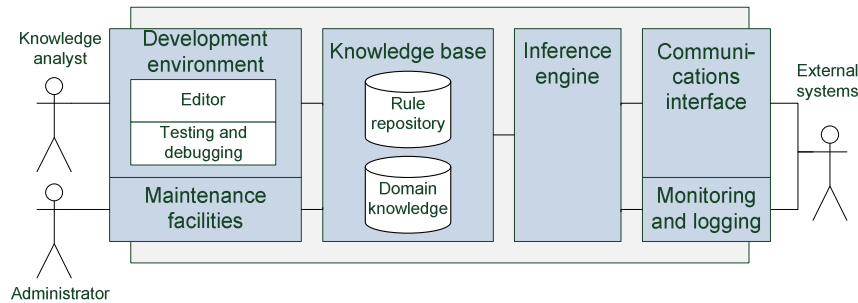


Figure 7. Architecture of a BRMS

business rules is therefore that business rules can be assigned to one or more activities in a business process.

3.3 Business Rules Management Systems

A number of commercial software systems exist that are marketed as Business Rules Management Systems (BRMS). Examples include (in alphabetic order) FairIsaac’s BlazeAdvisor, HaleyRules from HaleySystems, ILOG JRules and JBoss Rules from Red Hat. This section gives an overview of the architecture and functionality of a BRMS, as well as how they relate to other knowledge systems, in particularly Expert Systems (ES). Our analysis is inspired by the architecture of traditional knowledge systems, and loosely based on observations of existing BRMS products.

In existing literature, Graham has made an analysis of the architecture and functionality of four commercial BRMS products and describes his findings from a small case study implemented in each of these products [Gra06].

We did not find any definitions of a BRMS in existing literature. Therefore, in this thesis the following definition is used:

A Business Rules Management System (BRMS) is a domain-independent knowledge system that includes a repository, an execution mechanism and authoring and management functionality for business rules and is invoked by an external system through a generic communications interface.

The rest of this section explains how a BRMS has evolved from a traditional knowledge system, both from an architectural and an organizational standpoint.

3.3.1 Architecture

Figure 7 shows a reference architecture of a BRMS. By comparing this to Figure 2, the architectural differences with a hybrid expert system builder can be observed. For some components, alternative names exist but the “traditional” names have been used here when applicable in order to make the relation with expert systems

more clear. First a high-level comparison is made, and then the individual components are described.

From an architectural standpoint, the knowledge base and the inference engine are not very different from those of a (hybrid) expert system. The inference engine exposes its functionality through web services or Application Programming Interfaces (APIs). These replace the conversation-oriented user interface of an expert system. Instead of by a human knowledge user, the system is invoked by an external system, for instance a process server or an orchestration engine. The activities of the inference engine can be logged and monitored. The development environment and maintenance services both consist of a collection of facilities and are the result of what has begun as the expert system shell. However, it should be noted that one cannot draw a crisp boundary between elements that are inherently BRMS-specific and elements that are not. Finally, in large projects, multiple persons may fulfill a role as a knowledge analyst. Therefore, the administrator role has been included as an additional role.

As for the individual components, the following can be observed. The knowledge base employs multiple representation forms for declarative and procedural knowledge. In commercial products, domain knowledge is represented through an object model or a related form of knowledge representation. Production rules are used for the representation of business rules, sometimes in combination with other formats. Also certain procedural knowledge representation formats are often present [Gra06].

The inference engine uses a variety of algorithms to reason with the different forms of knowledge in the system, most of which originate from classic rule-based systems. The inference engine has an interface that is different from the user interface of an expert system. It is invoked by an external system which sends a message with a request to make a certain decision. From the start, the inference engine is fed with the information required to make this decision. This is done by attaching a number of objects to the message, which are placed in the short-term memory of the inference engine. If some information is missing, the engine must either send a message back to the calling entity or log an error. By contrast, the inference engine in an expert system starts with no information and incrementally adds small pieces of information to its short-term memory by asking questions to the user. The earlier mentioned external system that invokes the inference engine, may be a process server or an SOA orchestration engine. On invocation of the inference engine, it waits for the engine to finish and return the conclusion. The calling entity then continues and the inference engine turns idle.

When the system is in production, there is no human directly controlling it. In order to still keep a track record of the activities and possible errors of the inference engine, monitoring and logging facilities are needed, which can be used for Business Activity Monitoring (BAM) and maintenance.

The development environment and the maintenance services are meant to be used within the process of knowledge system development from the modeling phase and onwards. In existing products, often attempts are made to present the knowledge stored in the system in a less technical format such that it is understandable for all users that are involved with this knowledge [Gra06]. This

may be done with graphical models and schemes and language parsers in order to display and modify production rules in a way as close as possible to natural language. Debugging, testing and simulation facilities can be considered as essential components of the development environment. Analyzing a rule trace is an example of a debugging activity. Testing can be done by defining an expected output on a number of test cases in terms of rules fired and comparing actual runs with expected output. Additionally, a modified knowledge base can be deployed into a simulation environment. The goal of simulation is to discover the effects of modifications in the BRMS on other systems used by the organization and to remove potential failures before applying the system into a production environment.

Maintenance services include analysis and reporting for business rules, versioning and user management. Versioning and user management essentially add meta-knowledge to the knowledge base. With versioning, historical versions of the knowledge base are preserved and deployed versions are separated from the ones that are not ready yet. With user management, multiple people with different roles can access a BRMS in a different way. For example, the domain knowledge model may be constructed by one person, while a second person maintains the rule repository. A third person may perform a number of tests when something changes in the rule repository and deploy the system into production. At the same time, there may be people who need read-access to the rules but are not authorized to make changes, or who may only change specific parts of the knowledge base. This requires a (role-based) authentication and security model, which can be maintained by an administrator.

3.3.2 Comparison of BRMS and expert system applications

The architectural similarities and differences between a BRMS and an expert system have been discussed above. A comparison can also be made on the functional level. A number of similarities and differences related to practical usage are presented below. Some interesting trends can be seen, although it should be noted that most of the material that will follow is based merely on observations rather than extensive studies.

Durkin gives an overview of the major application areas for expert systems and mentions diagnosing, interpretation of data, planning, prediction and recommendation as the major problem types that are solved with an expert system. Major application areas are medicine, manufacturing and business [Dur94]. A web search for press releases and success stories of BRMS applications reveals that banking and insurance companies and governmental institutions are currently the main customers of BRMS vendors, and the systems are mainly used in administrative processes for activities like order processing, creditworthiness analysis and risk analysis.

Most commercial BRMS products are optimized to process high volumes of transactions, up to thousands per second. Many vendors mention speed as a major selling point of their products. By contrast, an expert system traditionally has a user interface through which a conversation is held with a human user. Therefore, one “transaction” in an expert system can last for a couple of minutes, so speed and concurrency are far less an issue.

There are also a number of features that are often associated with expert systems, but are rarely encountered in a BRMS. For example, the major BRMS products in the market today have no or little built-in support for uncertainty management or fuzzy inference [Hal01, Gra06]. This makes them hard to use in situations with a lot of uncertainty or vagueness, for example medical diagnosis, which is regarded as a classic expert system application area [Dur94]. Furthermore, a BRMS does not use an explanation facility in the same way as an expert system. Textbooks on expert systems frequently mention the importance of such a facility. The reason for this has already been pointed out in section 2.2.1. Nowadays, the acceptance of an “intelligent” computer system in an organization is far less an issue than it was three decades ago. But an even more important point is that a BRMS is not directly used by a human, but rather by external systems, which would question the decisions of the BRMS. There may be a whole chain of systems between the human end user and the BRMS, and the end user might not even know that at a certain point in a process a BRMS is used. The only situations where explanation is required (such as rule traces and access to the working memory or the goal agenda), is during development and testing of the knowledge base.

Another interesting point can be observed in the use of (natural) language by a system in an expert system versus a BRMS. Often it was considered important that a dialogue between a human and an expert system could take place in a form as close as possible to natural language. The developer, on the other hand, was supposed to be familiar with the “technical” representation of the knowledge in the system in the form of rules, objects and so on. He would add verbal constructs to the knowledge in the system of the system to make a dialogue possible. In a BRMS, by contrast, the use of language is also considered important, but it is used in quite a different way. The role of language in the dialogue between a BRMS and an external system is minimal; often standardized communication protocols such as SOAP are used. However, language plays an important role in the construction and preservation of knowledge in the knowledge base: a development environment of a BRMS uses language as a main representation format to present knowledge to a developer. To summarize, the use of language has shifted from the user interface to the development interface.

Von Halle [Hal01] and Graham [Gra06] give even more differences, but since they disagree on many points, not all of their findings are included here. For example, Von Halle argues that an expert system is used in domains where highly specialized knowledge is involved and a lot of inference rules are used, and a BRMS generally stores knowledge about a broader domain and uses relatively few inference rules [Hal01]. Graham disagrees on that based on personal experience [Gra06]. Nevertheless, it would be an interesting research project to investigate what kinds of problems for expert systems could be solved by a BRMS and vice versa, and if there would be significant differences in these applications with respect to the proportion of objects versus rules, the kinds of rules used (e.g. heuristic, recommendation, cause-and-effect etc.), the number of inferences and so on and what consequences this would have. For example, looking at the application areas it seems that heuristic rules are used more often in expert systems than in BRMS, but additional research has to verify this.

3.3.3 Organizational aspects

During the development and maintenance of a BRMS, different organizational roles can be distinguished. Schreiber et al. [S⁺99] describe a number of organizational roles that are distinguished in knowledge engineering and management. In the case of a BRMS this is not much different, but a number of refinements can be made. The roles identified by Schreiber et al. are the following:

- The *knowledge provider* has expertise on a particular domain. Traditionally, the knowledge provider is associated with an “expert” who has highly specialized knowledge in a certain field, such as medicine. However, often the knowledge inside a BRMS does not have to be highly specialized but represents criteria used in decisions that would otherwise been taken by for example an administrative employee or a business manager.
- The *knowledge analyst* (also sometimes called *knowledge engineer* or *rule analyst*) elicits knowledge from a knowledge provider or other sources and constructs models of this knowledge.
- The *knowledge system developer* is traditionally seen as a programmer who implements the knowledge models into a system and therefore forms the link between the knowledge analyst and the knowledge system. Traditional expert systems that did not have a shell, indeed required the knowledge system developer to use his programming skills in order to enter the knowledge in the system. However in a modern knowledge system and especially in a BRMS the role of the knowledge system developer has changed. BRMS software includes a development environment in which knowledge models can be constructed. Internally these knowledge models are translated to an executable format. The knowledge system developer is therefore no longer needed as a link between the knowledge analyst and the knowledge system. Instead, he gets a different and more supportive role as his tasks include for example designing communication interfaces for external systems that use the BRMS.
- The *knowledge user* is the direct or indirect (human) user of the knowledge system, also called the *end user*. In a traditional expert system, the knowledge user directly communicated with the system through a conversation-oriented user interface. In case of a BRMS by contrast, the knowledge user can be a whole chain of systems away from the system. In completely automated processes, a human end user can even be difficult to identify at all.
- The *project manager* has the responsibility for the progress of the knowledge system project.
- The *knowledge manager* is the initiator of knowledge system projects. He formulates knowledge strategies at the business level and is not directly involved with knowledge system projects.

With BRMS software, the following additional roles can be distinguished:

- The *administrator* facilitates user management in the development environment of the BRMS and maintains different levels of read- and write-access to the repository of business rules.
- The *tester* has always been present in (knowledge) system development. However, for a long time testing has not been regarded as a separate discipline, but seen as just one of the tasks of a software developer. The emergence of specific methods and techniques for testing (e.g. automated testing, TMap [PVT99]) indicate that this view has changed and testing is increasingly seen as an activity distant from programming. In a BRMS, testing needs to be done both internally on the knowledge base to validate the knowledge that it contains, and externally to validate the communication between the BRMS and other systems.

Finally, it should be noted that the mentioned roles are not limited to the initial development period, but go on continuously after the deployment phase, as the contents of the knowledge base evolve over time, reflecting the decision criteria in an operational business.

3.3.4 Business Rules Management

Business Rules Management has been defined by Von Halle as “*a formal way of managing and automating an organization’s business rules so that the business behaves and evolves as its leaders intend.*” [Hal01]. Although the term is fairly new, this discipline is deeply rooted in knowledge management which has been an active subject of research for a number of decades. An interesting point is that many sources on business rules management do not mention any relation with knowledge management. Therefore, an important question is what factors have led to the emergence of BRM, and what exactly the differences are between knowledge management and business rules management.

Graham, Von Halle and others mention numerous developments that have contributed to the emergence of BRM. At a high level, most of these developments fall in either of the two categories mentioned below.

Firstly, complex business rules are often embedded into the business logic of the information systems that support the business. When this is the case, the rules are executed by a computer system, which acts like a black box for most business managers. In most cases, managers do not understand computer programming languages. As a result, they do not get insight in the specification of the rules, and (by implication) the criteria for operational decisions. This is undesirable, since managers are ought to be responsible for these decisions. In order to preserve knowledge and keep them in control, a software system should be a “white box” to managers.

Secondly, in recent decades, companies face an increasing amount of competition as a result of globalization and increasingly demanding customers. In order to survive, companies have to be able to respond to market changes in a timely manner, and preferably faster than their competitors do. On the contrary, the information systems that support the business are often difficult to modify at the short term. Modifying software often involves high costs and requires time-

consuming processes of requirements engineering, designing, implementing and testing. This conflicts with the goal of a business to be able to implement changes quickly, and becomes especially difficult because business goals are a moving target to align with.

In order to provide a solution for these problems, BRM proponents advocate that either a BRMS is used, or a “conventional” software system where special efforts have been made to preserve the understandability and to increase the adaptability of business rules.

Following our observations, it seems that BRM can be seen as a branch of knowledge management that focuses on business rules (which are a form of knowledge) and advocates the use of BRMS software as repositories and execution engines for business rules. Therefore, BRM is more closely related to knowledge engineering than knowledge management in general. However, there is one difference that must not be overlooked.

A BRMS can be used as a repository system for decision criteria, and thanks to its development facilities, this can be done in a more advanced way than what used to be possible. In traditional knowledge engineering, knowledge is first elicited, then modeled and finally transferred into a knowledge system which uses the knowledge. Now, knowledge is first elicited and then modeled *in the development environment of the BRMS*. Seen in the context of the knowledge engineering process, it means that the BRMS has extended its role from only the implementation phase to both the modeling and the implementation phase. No longer the knowledge system developer, but instead the BRMS translates the knowledge models from a format understandable by a human into an executable format. The BRMS preserves the link from the design structure of the knowledge models to their technical implementation. During maintenance, these knowledge models are reused when changes are made to the knowledge base.

This paradigm is also used in Business Process Management Systems (BPMS). A BPMS is used as a repository for business processes and serves both as a design and modeling tool and as an execution engine for business processes and workflows.

3.3.5 Closing words

In section 3.1.1 it has been explained that in the past decades, an evolution has taken place in software architectures. Starting with stand-alone and monolithic systems, during the years more and more aspects have been externalized from information systems: first data, later on user interfaces and then workflows. We have now come to a point where business rules can be externalized. Time will learn whether will this trend will continue or not, and in ten years we will know whether 2005–2015 will perhaps be known as the period in which business rules are externalized from applications.

4 Implementing Business Rules Management Systems

This section describes the implementation process for a BRMS from the initial strategy and scoping phase until the deployment and maintenance, and sums up the most important steps and actions in between. A number of existing methods and techniques that can be used throughout this process, are positioned in this process.

In this research the following approach is taken. First of all, the main activities in a BRMS implementation process have been identified. Then a literature study has taken place in order to find methods and techniques that are potentially usable during the whole implementation process or in a part of it. After this, a number of questions were still unanswered. For these questions a number of suggestions are proposed. The contributions were validated by interviewing a few specialists at Logica and by undertaking a case study, which is the subject of the next section.

The remainder of this section has the following outline. First the implementation process is placed in the larger context of organizational change management. The focus is on software and knowledge engineering, which can be a part of a change process. Then an overview is presented of some methods and techniques that can be used, and their role within the process. This overview is followed by a brief analysis on the suitability of the methods and techniques. Finally, some relevant aspects will be discussed that are not mentioned by the mentioned methods and techniques.

4.1 Context of a BRMS implementation

This subsection describes the larger context of a BRMS implementation from the viewpoint of an IT services provider.

The implementation of any software system does not stand on its own, but is done in order to accomplish a certain goal or execute a strategy. The continuous process of the development, implementation and management of changes in an organization is often seen as a cycle. The Deming-Shewhart “Plan-Do-Check-Act” quality cycle is a famous example, but many variations exist that are based on it. From the viewpoint of Logica, this process can best be described by six different phases. The BiSi (Business Integration / System Integration) methodology used at Logica contains a set of models for the management of a continuously changing organization, and uses the following phases [Hoo07]:

1. *Manage*. Formulate strategy and goals. An example of a possible strategy is: “Average turn-around time for order processing must be decreased with 20%” or “A modification of a business rule must be implemented within 10 days”.
2. *Diagnose*. Analyze current situation and identify problems and bottlenecks.

3. *Define*. Define and analyze solutions to problems and bottlenecks. Set up a business case and estimate costs, feasibility and requirements of a solution.
4. *Develop*. When there is managerial commitment to one solution, realize the solution. This includes setting up new business processes, developing and acquiring hard- and software and so on.
5. *Execute*. Implement the solution into the organization. Develop and execute a transition plan for the deployment of software, transfer of knowledge (training) and so on. Such a transition is often executed in multiple phases.
6. *Monitor*. Measure performance of the implemented solution. Based on observations, a new iteration of the cycle can be started.

Methodologies for Software Engineering, Business Process Management, Enterprise Architecture and Organization Change Management can be placed within this cycle.

4.1.1 The process of software engineering

Software engineering entails all aspects regarding the development of a software system. A software system can be an aspect of a defined solution and therefore some phases within the BiSi cycle, such as *Develop*, may involve software engineering.

A classic approach to software engineering is the so-called *waterfall method* [Roy70]. This method divides the software engineering process in the following activities that are executed sequentially: *strategy definition, information analysis, system design, implementation and testing and operation maintenance*¹.

Knowledge engineering and software engineering share many activities. This has to do with the fact that the boundaries between knowledge systems and conventional systems are often not very sharp, as many conventional systems also to some extent contain knowledge [S+99]. Therefore we consider knowledge engineering as a branch of software engineering.

In section 2.1 we described the activities in the knowledge engineering process: *strategy definition, knowledge acquisition, modeling and design, implementation, testing and maintenance*. It takes little imagination that this process can be mapped one-to-one onto the waterfall method: *knowledge acquisition* would overlap with *information analysis* and *modeling and design* with *system design*. The other activities have the same names.

Although the waterfall method is very well-known, it has a reputation of being too rigid and inflexible [S+99]. Modern software engineering methods are often based on a *spiral model* [Boe88], in which a system is initially developed small-scale, and

¹ In different versions of this method, the name and number of the activities vary.

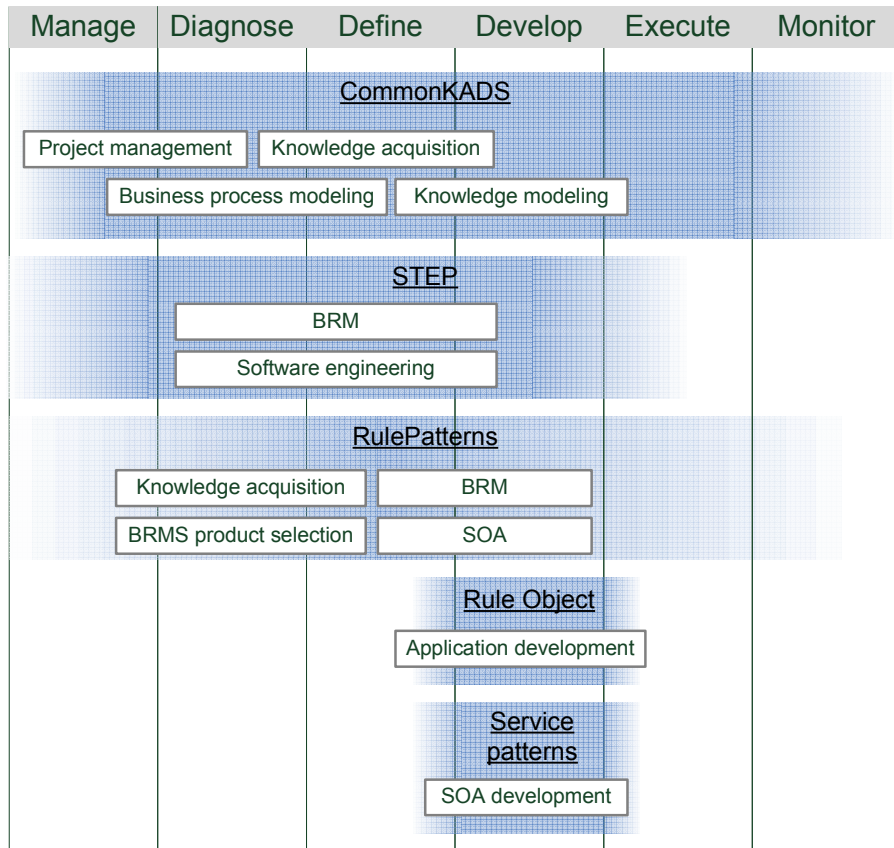


Figure 8. Methodologies and techniques positioned within the BiSi quality cycle.

grows by small steps, essentially by repeating phases from the waterfall method a number of times. Rational Unified Process (RUP) [Kru99] is an example of a well-known method for building large-scale software systems that has its roots in the spiral method.

4.2 A survey of methods and techniques

In this subsection an overview is presented of existing methods and techniques that are considered useful within the process of a BRMS implementation. Three of them, CommonKADS, STEP and RulePatterns are methods that span approximately the whole process. Two of them, Rule Object and the architectural patterns are techniques that focus specifically on technical design problems. The methods and techniques to be discussed, are:

1. CommonKADS, which is a well-established methodology for knowledge system development [S+99]. Although the terms “business rule” and “BRMS” cannot be found at a single place in the accompanying book, it turns out that this methodology is very suitable for certain phases in a BRMS implementation. The scope of this method spans from the strategy

definition until the development phase and includes knowledge acquisition and modeling.

2. The Business Rules Approach to Information Systems development by Von Halle [Hal01], from now on called “STEP” because of its four central principles. The scope of this method is approximately the same as CommonKADS.
3. RulePatterns by Graham [Gra06], a collection of activities and guidelines that can be followed during all mentioned phases in the knowledge engineering process.
4. Rule Object (not to be confused with RulePatterns!), a collection of patterns that can be used during the implementation phase, when business rules are implemented in conventional software rather than a BRMS [Ars01].
5. Service patterns, a small collection of patterns to implement knowledge-intensive activities as services in an SOA, to be used in the design phase.

The relation of these methods and techniques to the BiSi quality cycle can be seen in Figure 8.

4.2.1 CommonKADS

CommonKADS is a model-based approach to knowledge management and knowledge system development. It has been developed at the University of Amsterdam in The Netherlands in the past two decades and is recognized as one of the leading methodologies in its field. It has gradually evolved from KADS [BW92], its successor KADS-II and Components of Expertise [Ste90], and it has been used by universities and companies within the European Strategic Program on Research in Information Technology (ESPRIT) of the European Union. CommonKADS is an interdisciplinary methodology that, aside from knowledge management, incorporates elements from organization theory, business process management and information management.

The main principle of CommonKADS is that knowledge must be modeled before it is possible to use it in a knowledge system. Its model suite consists of six models that in turn consist of a number of worksheets or schemes. Because the full model suite is too comprehensive to be listed here, this thesis describes this methodology only in condensed form. The six models are:

Organization model

Constructing the organization model is done by filling in five worksheets in which the organization is described and various aspects of it are analyzed. This is first done from a high level. Then the relevant (i.e. knowledge-related) aspects are broken down incrementally in subsequent worksheets. In the first worksheet, the organization is described in general terms such as strategy, goals, value chain, vision and mission. Additionally, this worksheet contains a list of problems and opportunities the organization faces and “brainstormed” solutions for these problems. The second worksheet contains graphical or textual descriptions of the

organizational structure, business process(es), roles, knowledge assets and corporate culture. The third, fourth and fifth worksheet are the breakdowns of respectively the business process(es), the knowledge assets and the opportunities.

In the third worksheet the following characteristics are written down for each activity in the business process:

1. Performing agents, i.e. human roles or systems that execute the activity;
2. Location in the organization structure where the activity is executed;
3. Knowledge assets used by this activity;
4. Flag indicating whether the task is knowledge-intensive or not;
5. Indication of significance in terms of frequency, cost and so on.

For each knowledge asset, the following characteristics are written down in the fourth worksheet:

1. Agent that possesses this knowledge asset;
2. Activity it is used in;
3. Flags indicating whether or not the asset is used in the right form, the right place, the right time and the right quality, perhaps with comments.

In order to construct the fifth worksheet, a feasibility study must be conducted on the opportunities described in the first worksheet. In the fifth worksheet, the results of this feasibility study are decomposed into business, technical and project feasibility, along with the proposed actions to be taken.

Task model

The task model contains a further refinement of the third and fourth worksheet from the organization model. In the first worksheet of the task model, the following is described about each task:

1. Goal and value to the business;
2. Input tasks, output tasks, data flows and dependencies;
3. Input objects, output objects and internally used objects;
4. Timing and control aspects such as pre- and postconditions for the task;
5. Involved agents, knowledge assets and competences, resources and quality measures.

For points 2, 3 and 4, respectively a UML activity diagram, class diagram and state diagram can be used.

The second worksheet of the task model is a bottleneck analysis on knowledge assets. For each knowledge asset the following aspects are described, as well as whether this aspect forms a bottleneck.

1. Nature of knowledge (empirical, heuristic, specialized, uncertain, etc.);
2. Form of knowledge (mind, paper, electronic, action skill, etc.);
3. Limitations on availability (with respect to time, space, access, quality and form).

Agent model

The agent model is a refinement of the agents mentioned in the organization model. The CommonKADS definition of an agent is a human or computer system that executes a task in a business process and, optionally while communicating with the knowledge system. This model describes for instance the responsibilities and the used knowledge assets for each agent.

The information from the task and agent model are then combined into a coherent action plan in which:

1. the “to-be” and the “as-is” organization models are compared,
2. the changes in each activity and their impact on the business process and all stakeholders are described,
3. concrete actions for improvement are proposed.

This action plan is subject to managerial decision-making and approval.

Knowledge model

The knowledge model is a complete and implementation-agnostic specification of the knowledge use in a knowledge-intensive activity. The three subparts of this model constitute the *domain knowledge*, *inference knowledge* and *task knowledge* used. The domain knowledge model describes concept types and concept instances, relations between concept types and relations between values of concept attributes. The inference knowledge model describes the reasoning steps in an atomic and declarative way. This model thus contains the most low-level form of knowledge, described by an input-output specification (i.e. a black box). The task knowledge model decomposes a task as defined in the task model into a hierarchy of subtasks and a (procedural) description of how these subtasks are executed. This description may refer to inference knowledge or other subtasks.

When constructing this model, a high-level view is the starting point and from there detail is added incrementally. The authors mention numerous knowledge elicitation tips and guidelines, as well as a number of templates for various types of knowledge models.

Communication model

The communication model specifies the communications between agents in a knowledge-intensive activity. These agents may be a knowledge system and for example a user or another software system. First an overall communication plan is constructed, then the transaction flow (dialogue) between the agents is specified, and finally an exact specification of each information exchange between agents.

Design model

The design model is the link from the previous knowledge models to the technical implementation. The construction process for this model consists of four steps:

1. Design the system architecture for the knowledge system;
2. Decide on the target implementation platform;
3. Specify the architectural components of the system;
4. Specify the application within its architecture.

One of the main principles is that the final implementation should as much as possible preserve the information structure of the previous models. With the completion of the design model, the knowledge has been modeled in such a way that it is technically implementable. The choice for a certain target implementation platform affects how much of the design structure is preserved as well.

4.2.2 STEP

Von Halle's Business Rules Approach to Information Systems development is a comprehensive approach to software development with a central role for business rules. that uses six consecutive stages, which are divided into steps that in turn contain guidelines. Also the benefits and the distinguishing elements compared to other methods are addressed. She summarizes the main differences of her approach compared to other approaches to software development with the following four principles:

1. *Separate* rules from other aspects, both as requirements and implemented in software. This is a precondition for being able to modify them independently of other aspects of a software system.
2. *Trace* rules from their sources in business policies and regulations to their implementation in software systems. These links can be used to determine whether the implementation of a rule still corresponds to its specification. If a rule is implemented more than once, all places where it resides can be registered. Additionally, the impact of change in specifications on software systems can be predicted better.
3. *Externalize* rules, which entails making rules accessible to all relevant stakeholders. In practice, this means expressing them in natural language in order to make them understandable for managers.

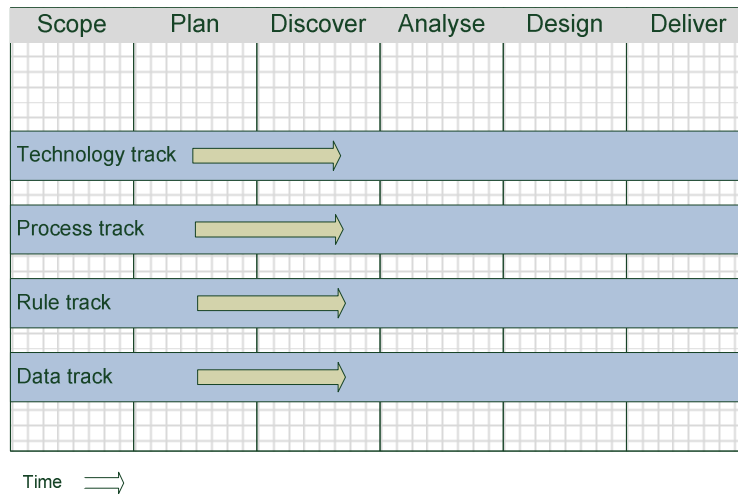


Figure 9. Von Halle's Business Rules Approach to application development.

4. *Position* rules for change. Because business rules are a volatile part of software systems, a software architecture is required that enables future changes to be implemented in a timely manner.

A schematic overview of the six stages is found in Figure 9. During development, four parallel points of attention (tracks) are distinguished: *technology*, *process*, *data* and *rule*. It should be noted that *process* means not only the business process but also the execution flow of activities in a system. The rule track is essentially the additional aspect that distinguishes this method from other software development methods. An important point made is that the rule track is present during all stages of development from the beginning until the end. The author also mentions that a dedicated software product for storing and executing business rules can be used, but this is always presented as optional.

The “Scoping” phase is the initialization of the project in which high-level requirements are gathered. In the next phase, “Planning”, a project plan is made in which the deliverables for all subsequent phases are established. The goal of the “Discovery” phase is to identify relevant concepts, business events, rules and other information through facilitated sessions and by analyzing descriptions and other written material. During “Analysis” logical rule, data and process models are established. Establishing a logical rule model entails grouping rules into coherent, technology-independent sets. In the “Design” phase, the process, data and rule models are integrated and implemented on the chosen technology platform. Rule management is also an aspect of design. This includes establishing a rule repository to make rule changes possible after delivery.

4.2.3 RulePatterns

Graham presents 42 practices that can be used throughout the system development process [Gra06]. This methodology is more recent than the two previous and therefore the author may have been aware of the possibilities and restrictions of current BRMS software. The practices are called *patterns*, although

this word is used in a different way than usual (in for example “*workflow patterns*”). Most patterns are not completely new, but borrowed from other disciplines such as project management and (conventional) software engineering. The patterns are classified into four categories:

1. *Patterns for requirements, process and architecture* (10 patterns); contains software development and project management patterns that may also be applicable in other contexts.
2. *Patterns for finding, writing and organizing business rules* (14 patterns); contains patterns that may be applied when it has already been determined what the *source* of the rules are; this may be a combination of program code, written documentation and expert knowledge (in this case, the business manager who takes the decisions is the expert).
3. *Patterns for knowledge elicitation* (14 patterns); contains patterns for the extraction of knowledge from human and written sources.
4. *Patterns for product selection and application development* (4 patterns); contains patterns involving organizational aspects of business rule management.

4.2.4 Rule Object

Rule Object is “*a pattern language for adaptive and scalable business rule construction*” by Arsanjani [Ars01]. Whereas the methods mentioned earlier are meant to be applied during the whole process of system development, Rule Object focuses only on the technical implementation of business rules into a software application. The author presents a total of 24 implementation solutions, of which 7 are elaborated on and the others are only mentioned briefly. These 7 describe different ways to organize business rules in application source code in such a way that the volatile parts (i.e. the decisions) are separated from the non-volatile parts (i.e. other business logic). The patterns vary in complexity and range from single methods to multiple class constructs.

Three of the described patterns are:

1. *Scattered conditions and actions*: no effort is taken to separate business rules from other parts of source code. Therefore, during maintenance it can be hard to identify the business rules in the code.
2. *Rule method*: a business rule is implemented in a method that returns a Boolean value indicating whether the rule applied or not.
3. *Rule object*: a business rule has its own class with at least three methods:
 - `assess()` contains the conditions of the business rule and returns a Boolean value indicating whether they apply or not.
 - `action()` contains the consequents of a business rule.
 - `apply()` executes the business rule. It first tests the conditions by calling `assess()` and then calls `action()` if the conditions evaluate to true.

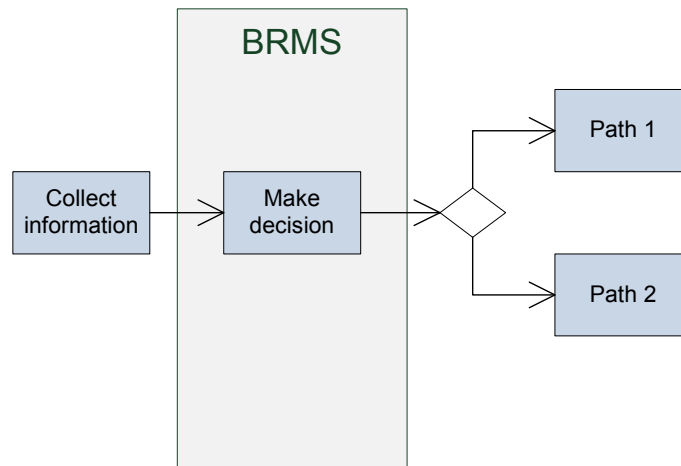


Figure 10. The *classification* pattern.

In subsequent patterns, layers of complexity are added to *Rule object*. This starts with externalizing the conditions and actions to separate classes. This offers greater flexibility and enables a dynamic binding of conditions and actions to rule constructs. In more complex patterns, other abstractions are proposed such as class hierarchies for multiple types of conditions and actions, and other facilities.

4.2.5 Service patterns

Most current BRMS software expose their decision-making functionality through web services. Among SOA practitioners, the term *decision service* is used to describe a web service that executes a (complex) decision using a BRMS.

One of the practical problems in an SOA is how to design web services in such a way that application functionality is neatly distributed, organized and prepared for reuse. Because SOA is fairly new, this is still a topic of ongoing research [Lae08].

Two patterns for decision services are mentioned that have been discussed during a number of interviews with domain experts. These patterns are based on the ideas of Van Hooijdonk [Hoo08] and several others. As I am not aware of any practical applications of these patterns at the moment, their current status is experimental.

The patterns can be interpreted in two ways: as *service patterns* and as *process patterns*. The service pattern interpretation is obvious, since services are the building blocks of an SOA. Since services at a high level should reflect the breakdown of a business process, services can be mapped onto business activities. Therefore, the process pattern interpretation is a logical consequence of a service pattern.

Pattern 1: Classification

Context: At a certain point in a business process, a knowledge-intensive decision has been identified. Based on the outcome, a certain branch path is followed in the process. A BRMS will be used to execute this decision, which is essentially a classification problem.

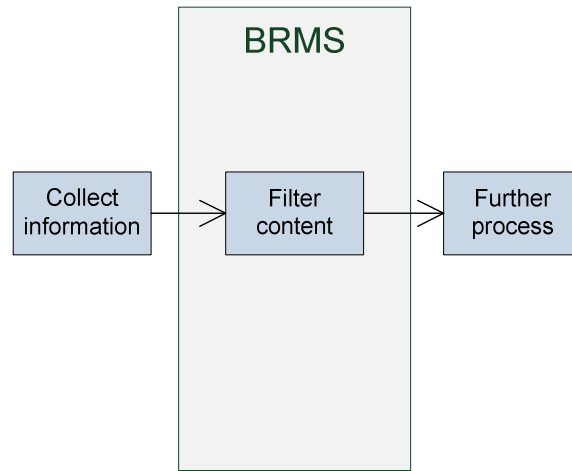


Figure 11. The *Content filter* pattern.

Problem: How do we make sure that our architecture conforms to the design principles of service-orientation such as separation of concerns and loose coupling?

Solution: Separate the activity of collecting data from the activity of evaluating the data. Do not involve the BRMS in the data collection, but only in making a decision based on it. Note that the diamond is only a notation format to describe a branch point in the process. It does not describe the activity of making a decision – this has already been done in the BRMS.

Pattern 2: Content filter

Context: In an organization, everybody has different rights to access certain kinds of information. Based on someone’s role, some information must not be disclosed and must therefore be filtered from a company dataset.

Problem: How can we design our architecture to prevent unauthorized access to restricted information?

Solution: First collect all information. Then use the BRMS as a content filter before further processing or displaying the information.

4.3 Analysis

In this subsection the previously mentioned methods and techniques are analyzed, based on their scope, emphasis and relevance, as well as a number of practical problems.

4.3.1 Analyzing the survey

CommonKADS is characterized by a “top-down” approach that starts with the identification of problems and opportunities, and then thoroughly analyses them by creating models that are extended in a number of steps. A major strength of CommonKADS is that the methodology also recognizes and covers the larger

(organizational) context of a knowledge system implementation. In the first worksheet of the organization model a strategy is defined. The other worksheets analyze the current business process and suggest potential solutions to current problems. All further models elaborate on the proposed solution subsequently in more detail. Therefore CommonKADS can be used during the *manage*, *diagnose*, *define* and *develop* phases of the earlier mentioned quality cycle. CommonKADS also covers project management aspects that can be used during the *execute* and *monitor* phases, but this is not dealt with in detail.

The STEP methodology as described in “Business Rules Applied” focuses on the software and knowledge engineering parts of an implementation process. We consider the STEP (separate rules, trace rules, externalize rules and position rules for change) philosophy which is reflected throughout the methodology as important. When a BRMS is used, complying to these principles is done naturally.

RulePatterns has a main drawback that it is a collection of concisely presented suggestions rather than a comprehensive methodology, and therefore has little value when used stand-alone. On the other hand, it contains a number of BRMS-specific guidelines, for example on the selection process of a BRMS vendor, which is not covered elsewhere.

On the topics in which CommonKADS, STEP and RulePatterns overlap each other, our preference goes out to CommonKADS. Not only has this methodology proven itself throughout the years, it also gives much attention to the business process. This methodology is compatible with the SOA practice to start with business process modeling and from there on “fill in” each service. STEP and RulePatterns finally include a number of guidelines that do not have to be taken into account explicitly, because they are automatically enforced by a BRMS.

Rule Object deals with object-oriented rule application development and can be considered out of scope for a BRMS implementation process. However, it may be possible to think of knowledge constructs that are hard to model inside a BRMS, which are easier to implement in a conventional programming language. In such cases, Rule Object can be used to separate rules from other business logic.

The discussed service patterns, to conclude with, are the most SOA-specific topic, in the sense that they focus on the design of (web) services, a subject that is not dealt with by methods such as CommonKADS or STEP.

4.3.2 Other considerations

A BRMS implementation imposes a number of necessary activities that have not yet been covered in detail. The following practical problems will be discussed briefly: assessing the need for a BRMS, selecting a product, classifying tasks suitable for a BRMS implementation and testing.

Assessing the need for a BRMS

The acquisition of a BRMS involves a considerable investment that consists of (for example) license costs and costs for training of employees. This investment is only made if the expected revenue on the short and the long term justifies it. A complicating factor is that this decision must be made relatively early in the

implementation process, before knowledge elicitation and modeling activities have taken place. Therefore the decision is based on indicators such as the amount of knowledge-intensive activities in a business process, the kind of knowledge and the expectation of the rate of changes to the business process and rules. These indicators are addressed extensively by CommonKADS.

Selecting a product

Once the necessity for a BRMS has been recognized, a product has to be selected. This decision can be quite complex, although decisions of this kind are certainly not unique. In fact, an organization may use a standard process that includes activities such as the submission of a Request for Information (RfI) and a Request for Proposal (RfP) to candidate suppliers. In other situations, a (company-wide) policy may limit the choice to certain vendors. Currently, there is few standardization between BRMS products. Every product offers slightly different algorithms and features for knowledge representation. This makes it even more important that a product is selected that offers the desired functionality. A Proof-of-Concept (PoC) implementation may therefore be made before making a final decision.

Classifying tasks for a BRMS

When a BRMS product has been selected and acquired, the implementation process eventually reaches the phase in which knowledge-intensive business tasks are modeled. Depending on the kind of knowledge-intensive task, either a BRMS or a conventional software system may be the appropriate implementation platform, or both. A number of considerations have to be taken into account when deciding on the implementation platform.

First of all, a BRMS may be preferred over a conventional software system because of its ability to represent knowledge in a much more understandable way compared to conventional programming languages. Additionally, knowledge is much easier to modify and maintain in a BRMS. However, for certain kinds of tasks a BRMS is not a good solution and a conventional software system may be preferred.

An important reason is that a BRMS fundamentally uses a declarative programming paradigm, which means that the knowledge specification is unordered. By contrast, procedural programming languages (such as Java) fundamentally use a procedural (top-to-bottom) programming paradigm. This means that tasks that can be represented in a format close to (for example) a decision table, are well-suited for a BRMS. Other tasks such as sorting, which is highly algorithmic, may be hard or impossible to represent in a purely declarative format. Therefore it is important to consider the nature of the task when deciding on the implementation platform.

Many BRMS products support *pattern matching* of objects to rules. A task that includes rules which require pattern matching may be a good candidate for a BRMS. An example of such a rule is: “If there are one or more *products* that have a *weight* above 1 kilogram, then add 10% to *delivery costs*”. This rule assumes a collection of *product* objects, in which the rule engine has to iterate over all of these objects to check the *weight* property. A conventional software implementation

would be more verbose because this iteration must be done “by hand”. Rules that use quantifiers like “one or more” and so on, are an indication that pattern matching can be used.

Furthermore, one might ask on what level of detail the decision for the implementation platform has to be made. The *task model* of CommonKADS may give guidance here. This model decomposes a knowledge-intensive task into one or more *task methods* that describe one part of a task in a pseudo-code-like notation. A task method is therefore more homogeneous than the task that it is part of. As a result, the task method seems at first sight the appropriate level of detail to base the decision on.

Finally, in an SOA it is irrelevant to the outside world (i.e. other services) how a service works internally, as long as the service complies to the agreed upon communication protocols. Therefore the decision for a certain implementation platform “under the hood” does not have influence on other services.

Testing

It goes without saying that testing is essential for the success of any IT system, and this is by no means different for a BRMS application. One of the nice properties of an SOA is that services can be tested as a black box in isolated fashion. Furthermore, a BRMS may include various facilities for testing and scenario analysis. Therefore, there are no excuses for bugs anymore.

5 Case study

Logica has been selected by a mortgage and insurance provider headquartered in the Netherlands as an outsourcing partner for its administrative processes. This company recognized that in order to maintain and strengthen its market position, it had to lower cost and shorten its time-to-market. It was expected that these requirements could be met if the majority of incoming mortgage requests were to be processed completely automatically. It turned out that this was not feasible with the existing back-office information systems and restructuring was required on the application and the infrastructure level.

Furthermore, it had been recognized that within its administrative business processes, certain activities were knowledge-intensive. These include for example assessing the creditworthiness of the client. The company suspects that the criteria used for these decisions could be formulated as interpretable business rules and evaluated by a BRMS.

In the case study, a small part of this business process has been implemented in a BRMS. Although the number of business rules involved is very small (only 6), it should be noted that the goal of this case study is to demonstrate certain methodologies instead of to implement a certain activity as complete as possible. The implementation, of which the basis has been laid here, can be extended relatively easily.

5.1 Background

The company identified that the trajectory starting at the arrival of a mortgage request and ending with the payment of the mortgage sum, involved three separate business processes, named *offer*, *acceptance* and *disbursal*, as shown in Figure 12. In the case study, the focus has been on the *offer* process.

The *offer* process starts when a mortgage request is received from an individual. First a number of checks is performed on the request, and then an evaluation takes place on the submitted data, resulting in either a decline or an offer. In case of a decline, the trajectory ends. In the other case, the offer is sent back to the client along with a number of conditions he has to meet. These conditions may include the submission of a life insurance policy, a bank guarantee and a National Mortgage Guarantee (Nationale Hypotheekgarantie, NHG). If the client agrees with and signs the offer, he has to send it back to the mortgage supplier. The receipt of a signed offer is the trigger for the *acceptance* process to start. In this process, the mortgage supplier examines the case and either rejects or approves it. A rejection again means the end of the trajectory, while an approval results in disbursal and closing, which is done in the *disbursal* process.

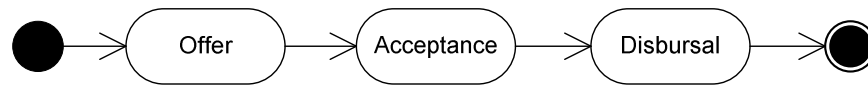


Figure 12. UML Activity diagram of the business processes involved in the supply of mortgages.

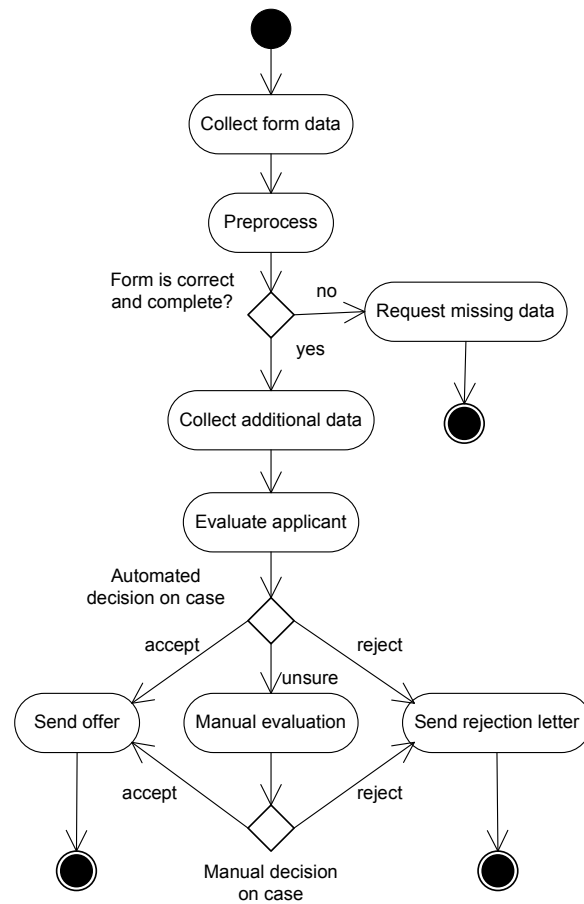


Figure 13. UML Activity diagram of the mortgage offer process.

In Figure 13, a simplified model of the future situation of the *offer* process can be found. From a high-level viewpoint, the three categories of activities in this process are data collection, evaluation and responding to the mortgage request. Initially, evaluation takes place automatically. However, in some cases the mortgage processing system does not accept or reject the request, but leaves the decision up to a human.

The process is initiated when a mortgage request is received from an applicant. This request has been sent by a broker and may be submitted electronically or on paper. A number of forms are attached to the request, in which the client gives

information about himself such as personal information, income, possible liabilities and details about the property that serves as the collateral.

In the first activity, *collect form data*, this information is read, and in the second activity, *preprocess*, a number of completeness and consistency checks are performed on the supplied information. If it passes through these checks, additional information is collected, otherwise, the client is requested to provide the missing information or to correct the mistakes.

The activity *collect additional data* involves the collection of data that the client did not supply directly, but which can be obtained from internal and external sources. Internal sources include a customer information system, in which it is checked whether the requestor is an existing customer and if true, which existing mortgages, insurance policies or investments he has. External sources include the Credit Registration Agency (Bureau Kredietregistratie, BKR) and the Mortgage Fraud Foundation (Stichting Fraudebestrijding Hypotheken, SFH). Respectively these checks return data about existing loans of the client and whether the client exists or not on a list of people that have committed mortgage fraud in the past. Furthermore, interest rates and closing costs are calculated. The interest rate also depends on whether NHG can be applied. The NHG guarantees payment of the residual debt to the mortgage provider when a client is forced to sell the property, but it fetches less than the amount of debt that was left in the mortgage. Because of this security, a lower interest rate can be charged.

The next activity, *evaluate applicant*, consists of evaluating the data that was prepared in the previous activity. The mortgage processing system creates a risk profile on the applicant to provide a recommendation on whether to accept a client or not. This risk profile consists of a borrower's loan capacity (which in turn is based on a combination of income, existing debt and education), the BKR and SFH results, investment behavior (risk-seeking, neutral or risk-averse) and his deposited equity. On each of the parts, a score is calculated. When each of the scores is higher than a certain cut-off value, the client "passes" on that part. The final recommendation depends on the parts and the total.

Depending on the recommendation, a mortgage request is either automatically accepted, automatically rejected, or, in unsure cases, prepared for manual processing. In the *manual evaluation* activity, a human will investigate the case, request additional information if necessary, and decide to accept or reject the request. In case of acceptance, an offer is generated and sent to the client. In case of a rejection, a rejection letter is sent. Here the process ends.

If the client accepts the offer, he has to provide a number of additional documents (like a notary signature) and send them back along with the signed offer to the mortgage provider. When the mortgage provider receives the documents, in the *acceptance* process these additional documents are processed and a final decision is made on whether to supply the mortgage or not. This process is manual for the most part; IT plays a supportive role by fulfilling tasks such as workflow and data management. If the company decides to supply the mortgage, the *disbursal* process is triggered to settle the contract and pay the mortgage sum.

5.1.1 Problem description

One of the tasks within the “Evaluate applicant” activity is the evaluation of historical information of the customer, which is obtained electronically by querying the database systems of the SFH and BKR. This information is used to identify potentially problematic customers. Based on this information, some applicants can immediately be rejected or prepared for manual evaluation. If no indicators for problems are found, the applicant is evaluated on other evaluation criteria.

A query to the SFH system for a particular person returns a Boolean indicating whether that person appears or not in the SFH database.

A query to the BKR system returns data about current and past credits that have been registered at the BKR. If the queried person did not have any outstanding credits in the last 5 years, the query returns that nothing has been found. If the person has had one or more outstanding credits in the last 5 years, the query returns personal data of the borrower and a list with all credits of the last 5 years. The following personal data is returned:

- A. Full name (including maiden name in case of a married woman)
- B. Initial(s)
- C. Date of birth
- D. Address
- E. Zipcode and City

The following data is returned about each credit:

- A. Amount
- B. Origination date
- C. End date according to contract with the borrower (if applicable)
- D. Real end date (if applicable)
- E. Type of credit (RK, AK, VK, HY, SR, RO or TC)
- F. Flags (A, H, 1, 2, 3 and 4) indicating any special situations (if applicable)

The meaning of the credit type indicators and the flags can be found in Appendix A.

Business rules

The evaluation of the mortgage applicant uses a number of business rules which had been formulated in an earlier stage by the mortgage provider. However, these had not yet been specified in a structured way or using a formal knowledge model, but were described in the following semi-structured way²:

- 1) If the applicant is in the SFH database, the decision is REJECT.
- 2) A certain number of codes or combination of codes immediately lead to a certain decision. This is specified in the following table:

Types or flags found	Decision
A or 1, with H	ACCEPT
A or 1, without H	UNSURE
SR or RO	REJECT
2, 3 or 4	REJECT

- 3) The decision is UNSURE if there is evidence that the applicant has repeatedly taken loans with the intent to repay debts from previous loans. This is assumed to be the case if the number of credits with type RK is more than one, and when they are sorted by starting date the following conditions are true:
 - a. For each adjacent pair of credits, the start date of the second credit minus the end date of the first credit is between 0 and 1 months.
 - b. For each adjacent pair of credits, the amount of the second credit is greater or equal than the amount of the first credit.
- 4) When the number of credits is greater than 4, the decision is also UNSURE.
- 5) The default decision is ACCEPT.

The rules that lead to a REJECT decision are preferred to the rules leading to UNSURE, which in turn are preferred to the rules that lead to ACCEPT. In other words, if both a rule leading to REJECT and to UNSURE apply, the decision is REJECT.

5.2 Methodology

In the case study, a decision service is created that generates a decision for further processing of an applicant, based on the evaluation of his SFH and BKR information. This SFH and BKR information is used as the inputs for this

² The information has been paraphrased for anonymity reasons.

decision service. The output is a decision for further processing: either “accept”, “reject” or “unsure” (which results in manual evaluation). This decision service has been implemented in the ILOG JRules BRMS.

5.3 The ILOG JRules BRMS

ILOG JRules is a Business Rules Management System developed by ILOG, a French company that is also known for CPLEX (a numerical optimization package), CP (Constraint Programming) and other industrial software [ILO08]. JRules originates from Rules, a collection of C++ libraries for object-oriented production system development that has been developed in the past 15 years. In the early beginnings, Rules was meant to add rule-based programming functionality to the C++ language and applied forward chaining rules on C++ objects, using the Rete pattern matching algorithm [Alb94]. In 1997, Rules was released also as a class library for Java [Flo97].

As we speak, JRules has evolved into a BRMS that contains all components as described in the reference architecture in Figure 6 in some or another way. The current version is 6.6 and has a quite impressive feature list. Therefore, in our discussion no attempt will be made to cover every single feature, but rather the elements that are of interest for this research, namely on the one hand its architecture compared to the reference architecture for BRMS and knowledge systems and on the other hand the development methods that can be applied when using JRules. The complete documentation of JRules can be consulted at <http://www.ilog.com/products/jrules>.

Its main components are:

- *Rule Execution Server*, which contains an execution unit (XU) with the inference engine, monitoring and logging facilities and interfaces to external systems. It also includes a web interface to the logging facility and a facility to view the statuses of the deployed ruleapps.
- *Rule Studio for Eclipse*, a plugin for the Eclipse IDE that functions as the development environment for business rule systems.
- *Rule Team Server*, an editing environment intended for “business users” to have access to the business rule system.
- *Rule Scenario Manager*, a testing environment for ruleapps.

When we compare this to Figure 6, from a high-level view the right half part of the BRMS picture has been brought together in Rule Execution Server, whereas Rule Studio, Rule Team Server and Rule Scenario Manager form the left half. The knowledge base has not been mentioned as a component yet. Actually, this is a collection of files that can reside in three separate (logical) places: first in Rule Studio when it is developed, in Rule Team Server when it is made accessible to business users and in Rule Execution Server when it is deployed.

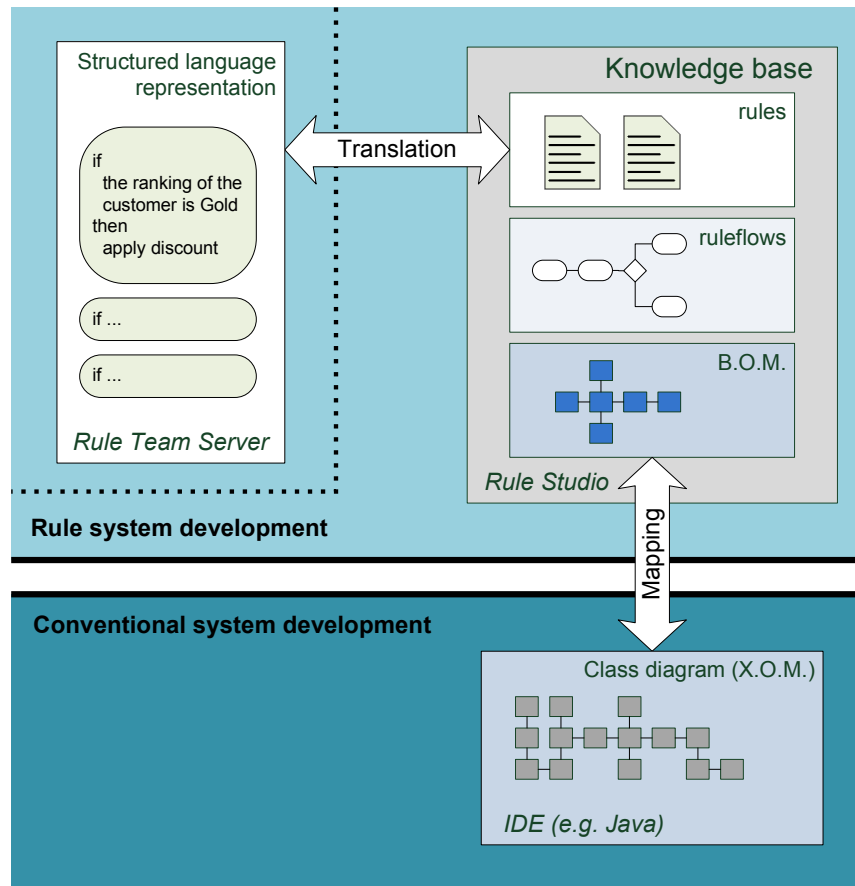


Figure 14. Knowledge representation and system development with JRules.

5.3.1 Knowledge representation in JRules

JRules employs three kinds of knowledge representation: objects, ruleflows and rulesets. It can hold multiple knowledge bases, just like a DBMS can hold multiple databases. A knowledge base as a whole is called a *rule project*. JRules takes a different approach to development of objects, ruleflows and rulesets, which is illustrated in Figure 14. Internally, the objects are stored in text format in a way that looks like a Java file with only interface definitions. Ruleflows and rulesets are internally represented by the ILOG Rule Language (IRL). This language also supports loops, branches and exceptions, as well as Java expressions.

The collection of objects in a rule project is called the Business Object Model (BOM). The objects in the BOM have properties and methods, and a vocabulary with linguistic descriptors of the objects, properties and methods. The BOM can be mapped onto an Executable Object Model (XOM), which represents the actual implementation of the objects in the BOM. This can be defined by a set of Java classes or by a web service description (WSDL). By maintaining this link, the business rule system can share (a subset of) the object model with conventional applications, and at the same time be implemented and maintained separately.

A ruleflow specifies in which order certain rulesets (collections of rules) are executed. This is displayed in a flowchart-like graphical notation. Ruleflows support both serial and parallel execution of rulesets through forks and joins, as well as branch points. A ruleflow starts when a *rule session* is initiated by an external application. This application has to pass a number of objects as input parameters. At the end of the session, a number of objects can be returned as output parameters.

Rulesets in turn can contain *action rules*, decision tables, decision trees and so-called *technical rules*, as well as a number of other items. An example of an action rule specified in IRL is:

```
rule late_payments {
  when {
    mypackage.customer() from customer;
    evaluate (customer.noLatePayments > 3);
  } then {
    result.message = "Too many late payments";
  }
}
```

To the user the action rules are usually not presented in IRL, but rather using the BOM vocabulary in a structured English-like language called Business Action Language (BAL). Action rules specified in BAL are based on the following basic construct:

```
set
  <definition(s)>
if <condition(s)>
  <action(s)>
else
  <action(s)>
```

The action rule specified earlier in IRL looks in BAL like the following:

```
if
  the number of late payments of 'the customer' is more
  than 3
then
  set the message of 'the result' to "Too many late
  payments"
```

Here 'the customer' refers to an object that will be passed to Rule Execution Server by the calling system as an input parameter. The number of late payments and the message are properties of respectively the Customer object and the Result object from the BOM and use its specified vocabulary. Terms like *if* and *is more than* are BAL constructs. It is possible to customize them, to create new constructs and even to create completely new languages using the Business Rule Language Definition Framework (BRLDF).

Although BAL is constructed to support the most frequently used rule constructs, it is possible to create “technical rules” that use only IRL and have no BAL

representation. Decision tables and decision trees are presented in a graphical editor that generates IRL for internal use.

Also metadata can be attached to rules in a ruleset such as priority, a version number, user defined properties, comments and authorizations for certain users or user groups to make changes. Finally, they can be queried using an English-like query language in which the mentioned metadata can be used as search criteria.

5.3.2 System development with JRules

The development lifecycle for a business rules system built with JRules is based on the notion that rulesets will change relatively often and the time needed to implement rule changes needs to be minimal, while the BOM and the ruleflows need to change less often compared to rulesets. Therefore, rule development and maintenance is done with Rule Team Server, while development of the BOM and ruleflows (and rules, if desired) are done with Rule Studio.

In Rule Team Server, rules sets can be composed. Action rules are composed by adding, selecting and entering building blocks (artifacts) using BAL. Decision trees and decision tables are edited using a graphical editor. This application is designed to be used by “business users”, which means in practice that people with no programming skills should be able to use the application. Read and write permissions can be set on the ruleset level to authorize or deny certain people or groups to change certain rulesets. Changes made to a rule project do not immediately affect systems in production. Instead, the changes must first be approved and deployed by an authorized user.

With Rule Studio, rule projects can be constructed. It contains editors for the BOM, ruleflows and rulesets, and includes facilities for debugging and synchronizing different versions of rule projects. Using the BOM import facility, a BOM is constructed by importing a Java class package or a WSDL file. It establishes a link from the imported classes (the XOM) and the BOM, and generates a default verbalization based on the class and method names. We can illustrate this by the following Java class:

```
public class Customer {

    private String rating;

    public String getRating() {
        return rating;
    }

    public void setRating(String rating) {
        this.rating = rating;
    }
}
```

The BOM entry generated by JRules would in this case consist of a *Customer* object with a *rating* property, and respectively the following (verbalized) getter and setter:

```
the rating of the customer
set the rating of the customer to ...
```

Both the mapping and the verbalization can be customized. For instance, it is possible to create a BAL construct for the getter and setter if some form of complex mapping is required (for instance, if another function in the XOM has to be called).

When a rule project is ready for deployment, the interface to external applications must be constructed. JRules provides a web service generator if the project is to be deployed in an SOA environment. Alternatively, a Java program can use the class library of the Rule Execution Server to invoke rule project directly.

5.3.3 Rule execution

The rule engine in Rule Execution Server support supports algorithms for rule execution: *sequential*, *fastpath* and *reteplus*. The algorithm to be applied can be configured per ruleset. By default, *reteplus* is used, which uses forward chaining, Rete-based pattern matching and conflict resolution. The techniques used for conflict resolution are refraction (a fired rule cannot fire again unless a “new fact” occurs), priority of rules and recency of matched objects. The engine can be configured either to stop processing a ruleset when one rule has fired, or to continue with the other rules on the agenda. If necessary, execution can be fine-tuned by giving specific commands with IRL to each of the mentioned algorithms. The used algorithm affects not only the execution speed, but the execution result. Therefore an algorithm may be chosen based on the structure of the rules and objects and the desired goals. In the documentation of JRules, the three algorithms are thoroughly explained and therefore we refrain from discussing them here.

5.4 Case study implementation

The problem of evaluating an applicant’s credit history as described in section 5.1.1 has been implemented as a rule project in JRules. This problem consists of the two subtasks of deciding on an applicant’s history as recorded in respectively the SFH and BKR databases.

The inputs to this decision service are an *Applicant* object and a collection of *Credit* objects. The output is a *Decision* object. The properties of these objects are shown in Figure 15. Figure 16 shows that the ruleflow of the decision service consists of two rulesets. The first ruleset, *SFH check*, has only 1 business rule which checks whether the Boolean *fraudulent* of the *Applicant* object has a value of *true*. The *BKR check* ruleset has 5 business rules. Both rulesets are tuned in such a way that the rule engine must stop executing rules as soon as one rule has fired. The rules that lead to a REJECT decision were given a priority of *high*, whereas the rules that lead to UNSURE have normal priority. The “default” rule to ACCEPT an applicant has *lowest* priority, because it only applies if none of the other rules apply. The BAL representation of the rules in both rulesets is shown below.

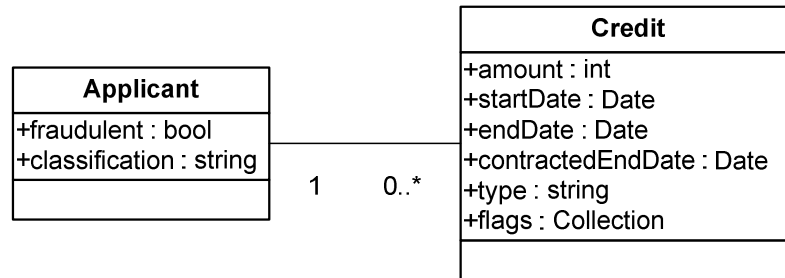


Figure 15. Domain knowledge model for the case study application.

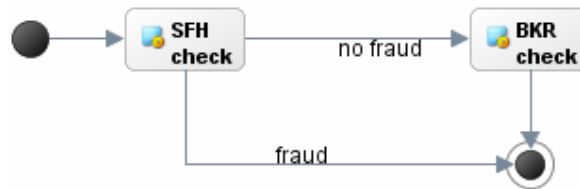


Figure 16. Ruleflow of the SFH and BKR check tasks.

Rule package “SFH check”

Business rule 1: Fraudulent applicant

if

'the applicant' has committed fraud

then

set the classification of 'the applicant' to "REJECT" ;

(Priority: high)

Rule package “BKR check”

Business rule 2: Previous late payments

definitions

set 'credits with late payments' to all credits in the credits of 'the applicant'

where the flags of each credit contain "A"

or the flags of each credit contain "1" ;

if

there is at least one credit in 'credits with late payments'

where the codes of this credit do not contain "H" ,

then

set the classification of 'the applicant' to "UNSURE" ;

Business rule 3: Maximum number of credits

if

there are more than 4 credits in the credits of 'the applicant'

then

set the classification of **'the applicant'** to **"UNSURE"** ;

Business rule 4: Bad debtors

definitions

set **'one of the applicant-s credits'** to a **credit** in the credits of **'the applicant'** ;

set **'the credit codes'** to the codes of **'one of the applicant-s credits'**;

if

any of the following conditions is true :

- **'the credit codes'** contain **"2"**
- **'the credit codes'** contain **"3"**
- **'the credit codes'** contain **"4"**
- the type of **'one of the applicant-s credits'** is **"SR"**
- the type of **'one of the applicant-s credits'** is **"RO"** ,

then

set the classification of **'the applicant'** to **"REJECT"** ;

(Priority: high)

Business rule 5: Sequential credits

definitions

set **'the first credit'** to a **credit** in the credits of **'the applicant'**

where the type of **this credit** is **"RK"** ;

set **'the second credit'** to a **credit** in the credits of **'the applicant'**

where the type of **this credit** is **"RK"** ;

if

all of the following conditions are true :

- the start date of **'the second credit'** is after the start date of **'the first credit'**
- the start date of **'the second credit'** is after or the same as the end date of **'the first credit'**
- the amount of **'the second credit'** is more than the amount of **'the first credit'** ,

then

set the classification of **'the applicant'** to **"UNSURE"** ;

Business rule 6: Default decision

then

set the classification of **'the applicant'** to **"ACCEPT"** ;

(Priority: lowest)

Business rule 1 is self-describing. Business rule 2 is the implementation of the second row in the decision table of section 5.1.1. Business rule 3 checks whether the number of credits of the applicant does not exceed the given maximum. In business rule 4, all possibilities that lead to a REJECT decision are evaluated. In rule 5, it is detected whether the applicant has multiple increasing debts as described in section 5.2. For this rule, the engine has to apply pattern matching on all pairs of credits of type RK. Finally, when no other rule fires, the decision is ACCEPT. Because rule 6 does not have an if clause, the consequent always fires. Because it has *lowest* priority, it is the last rule scheduled on the agenda

5.5 Discussion

In the previous subsection it has been demonstrated how knowledge can be modeled and entered in a BRMS. A proof-of-concept has been given of one specific task “in the middle” of an BRMS implementation process. Everything that comes before the modeling activity, such as strategy formulation and knowledge elicitation, had been done earlier. Therefore, the business rules described in section 5.1.1 were the starting point for our demonstration. A few remarks can be made on the implementation.

Firstly, the structured English notation of the business rules using BAL is understandable, and the shown rulesets give a good first impression of the capabilities of the notation language. In business rule 3 it is illustrated that the language supports unsorted lists and existential quantification, like:

```
IF there are one/two/... <object(s)> in <collection>
```

However, the language does not support *sorted* lists. Neither does it support universal quantification, as in:

```
IF for all objects in <collection> holds that <condition>
```

Readability decreases with the length of the rules, as shown in business rule 5. To keep the business rule repository maintainable, splitting rules could be considered when possible.

Secondly, it should be noted that JRules does not support constraints like “ $X < 3$ ” in the consequent of a rule, for example:

```
IF A < x THEN B < y
```

From a technical standpoint this might be no surprise, because evaluating a ruleset with multiple constraints in this form would require constraint-solving algorithms, which is not really considered as a core feature of a rule-based system. Constraints must therefore be written in an “IF...THEN” rule format, if this is possible.

Thirdly, JRules guides the developer into a “top-down” approach to knowledge modeling: it lets the developer start with a domain knowledge model (BOM). From there on the inputs and outputs to the decision service are defined, followed by the ruleset and ruleflow definitions. Only when these “high-level models” have been constructed, the developer is able to formulate business rules. This methodology is contrary to the approach of a product such as HaleyRules, which enables the developer to write rules first and from there “reverse-engineer” to an object model [Gra06]. We do not however regard this restriction of JRules as a disadvantage. Of course, once the business rules have been defined, it remains possible to make changes to ruleflows and (to a certain extent) to the BOM.

To conclude, the development functionality of JRules allows it to be used as a knowledge modeling and implementation environment. JRules clearly complies to the STEP principles discussed in chapter 4, because it stores and organizes business rules into decision services that are separated from other functionality.

Due to their language representation, insight in the criteria of operational decisions is improved, which is exactly the goal of a Business Rules Management System.

6 Concluding remarks

In the previous chapters we thoroughly explained the technological and methodological aspects of Business Rules Management Systems. Additionally, we have placed them in perspective by looking at their history, applications and organizational context. This chapter concisely sums up our findings and addresses a few unsolved problems.

6.1 Conclusions

Hereby we present our answers to the research questions posed in section 1.3.

What is the relation between Business Rules Management Systems and Expert Systems?

The inner core of a BRMS can be considered as a hybrid expert system builder. As technology and insights have evolved, the features of a typical BRMS exceed those of a typical expert system. Apart from a number of supportive elements of current BRMS products that are not traditionally associated with expert systems, such as user management, activity monitoring and facilities for automated testing, we believe that a BRMS has two major differences compared to a traditional expert system builder.

Firstly, an operational BRMS is not directly controlled by a human, but is meant to be used within a larger software system suite that is often organized according to the principles and technologies of Service-oriented Architectures. As a result, the user interface to a BRMS is no longer oriented at having conversations with a human, but uses SOA technologies (web services) to accept requests and to send responses to other systems.

Secondly, the development environment of a BRMS has knowledge modeling facilities and partially automate the implementation phase of knowledge engineering. These facilities enable the BRMS to be used not only as a repository system and inference engine, but also as a modeling environment for business rules and other forms of knowledge. The BRMS creates and preserves a link from the knowledge in modeled form to a machine-executable format.

These differences are most notable in the use of language at both “sides” of the systems: the interfaces to respectively the knowledge user and the knowledge system developer. A traditional expert system uses language to have conversations with a human knowledge user. The knowledge system developer would add language elements to the knowledge model implementation to make these conversations possible. In a BRMS, by contrast, language is used primarily in the development environment to ease the construction of the knowledge base. For communication with a knowledge user, which is an external system, a messaging format such as SOAP is used.

During the development and maintenance of a BRMS, different organizational roles can be distinguished. Typical roles associated with traditional expert systems are the knowledge provider (expert), knowledge analyst, knowledge system

developer and knowledge user. These roles also apply to a BRMS, but with some minor differences. Firstly, the BRMS is not directly operated by a knowledge user, but via a number of intermediary systems. Secondly, due to its development and modeling facilities the distinction between the knowledge analyst and knowledge system developer becomes less crisp. Finally the development environment facilitates a number of additional roles such as knowledge administrator and tester.

What is the relation between the concepts Business Process Management and Business Rules Management?

Business rules are a way of expressing the criteria for operational business decisions. These decisions are part of activities in a business process. Business rules therefore are a way of expressing the composition of knowledge-intensive activities and determine the further course of a case in a business process.

Service-oriented Architectures form a link between BPM and BRM. It has been considered good practice to organize services in an SOA in such a way that they reflect the underlying business process. This means that business process modeling is a prerequisite for the design of an SOA. In a technical infrastructure of an SOA, activities of the business process are explicitly modeled in a distinct layer “above” the technical services that are exposed by the information systems implementing these activities. A BRMS is such an information system, which exposes its functionality through *decision services* to other services. BRM, which includes knowledge modeling, is therefore a part of the design of these decision services.

Which methods can be used to implement a Business Rules Management System in a Service-oriented Architecture, and how can these methods be combined?

The context of a BRMS implementation is the process of implementing a business strategy. One way to describe this process is to divide it into the following phases: define strategy and goals (manage), analyze current situation (diagnose), define, develop and execute a proposed solution and monitor the performance of the implemented solution.

Within this process, different methodologies and techniques can be used: CommonKADS, STEP and RulePatterns are meant as methodologies for the whole implementation process, while Rule Object and two service patterns deal with specific technical implementation problems.

Because of its targeted and sound approach, fitness for our goals and established history, we recommend the CommonKADS methodology for the first four phases of the implementation process (define strategy, analyze situation, define solution and develop solution). Although the scope of CommonKADS is approximately the whole process, it particularly deals with these phases and is not very specific at for example technical and platform-specific issues. When used in the first four phases, the result afterwards is a number of models in which knowledge is modeled.

The development environment of a BRMS includes knowledge modeling facilities. Therefore, as soon as the knowledge models have been constructed, they can be directly translated to the BRMS development environment. Alternatively, the BRMS can be used as the primary modeling environment to construct the knowledge models, although it has to be kept in mind that a BRMS focuses only on specific kinds of knowledge (particularly rules).

Although CommonKADS predates SOA, we are convinced that it is compatible with the SOA paradigm because it shares with SOA its strong orientation on the business process. Nevertheless, the methodology does not provide any help on SOA-specific tasks such as service design. We have tried to fill this gap by proposing two patterns that can be used as service or process patterns for two specific BRMS tasks: classification and content filtering.

Finally, we have briefly addressed a number of practical problems related to the use of a BRMS. An organization considering to use a BRMS needs to be aware of the differences between products with respect to kinds of knowledge representation supported, inference algorithms and so on. Therefore, it should undertake studies to examine the suitability of a BRMS for a particular task. Both theoretical analysis and practical tests by means of a proof-of-concept should precede the acquisition of a product. To a certain degree, expertise in artificial intelligence and knowledge representation is needed for these analyses.

6.2 Suggestions for further research

A couple of questions are still unanswered and therefore are the subject of further research.

First of all, it would be interesting to see whether the applications of expert systems versus a BRMS have different characteristics. The majority of BRMS applications that were encountered during this research are in administrative processes. The question is therefore whether this phenomenon can be attributed to problem characteristics, or has a different cause. In other words: do problems in administrative processes have any properties that can explain the success of a BRMS in this application area?

Secondly, two patterns for service design have been described: *classification* and *content filter*. There are no established standards or guidelines for the composition of services. Therefore it would be interesting to see practical applications of these patterns. Furthermore, the application of a BRMS for a different type of problem may impose a need for other patterns. It is to be expected that the extension of the proposed pattern collection has considerable practical relevance.

Glossary

Term	Explanation
BAL	Business Action Language; a notation format used by JRules to describe business rules in an understandable way.
BOM	Business Object Model; a conceptual model containing the objects relevant to the business. Used by JRules to describe a domain knowledge model.
BPM	Business Process Management; describes methods, techniques, and tools to support the design, enactment, management and analysis of operational business processes [AHW03].
BRG	The Business Rules Group, a organization of system and business analysis methodologists; has published a number of papers on business rules.
BRM	Business Rules Management; a formal way of managing and automating an organization's business rules [Hal01].
BRMS	Business Rules Management System; a domain-independent knowledge system that includes a repository, an execution mechanism and authoring and management functionality for business rules and is invoked by an external system through a generic communications interface.
business process	a collection of activities that are executed in a predefined order and contribute to a clear business goal.
business rule	a statement that expresses a criterion for an operational business decision.
CommonKADS	a methodology for knowledge engineering and management.
expert system	a knowledge system that emulates a human expert.
inference engine	component of a knowledge system that uses AI techniques to decide on a problem.
IRL	Internal Rule Language; used by JRules.
JRules	a commercial BRMS by ILOG.
knowledge base	component of a knowledge system in which a domain knowledge model is stored; also called <i>long-term memory</i> .

knowledge system	a software system in which knowledge is explicitly represented in a knowledge base.
Rule Object	a collection of patterns for business rule construction in software applications.
RulePatterns	a methodology for information system development with a central role for business rules.
service	a (software) component that performs a clearly defined task and can interoperate with other services.
SOA	Service-oriented Architecture; a paradigm for the design of an (IT-)infrastructure that consists of autonomous and loosely coupled components (called services) that individually perform a clearly defined and unique task and work together seamlessly and consistent with business objectives.
SOAP	Simple Object Access Protocol; a messaging format used in web services technology.
STEP	a methodology for information system development with a central role for business rules.
Web services	a commonly used technology platform for the implementation of an SOA.
WSDL	Web Service Description Language; a Web Services standard for the description of web services.

References

- (AH00) Van der Aalst, W. M. P. and Van Hee, K. (2000). *Workflow Management: Models, methods and systems*. MIT Press Ltd., ISBN: 0262720469.
- (AHW03) Van der Aalst, W. M. P., Ter Hofstede, A. H. M. and Weske, M. (2003). Business Process Management: A Survey. In *Lecture Notes in Computer Science*. Vol. 2678, pp. 1-12.
- (Alb94) Albert, P. (1994). ILog Rules, embedding rules in C++: Results and limits. In *Proceedings of the OOPSLA'94 Workshop on Embedded Object-Oriented Production Systems (EOOPS)*. Technical report LAFORIA 94/24, Institut Blaise Pascal.
- (Ars01) Arsanjani, A. (2001). Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction. In *Proceedings of the 8th Conference on Pattern Languages of Programs (PLoP)*, pp. 370-402. IEEE Computer Society, Washington DC, USA.
- (Boe88) Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*. Vol. 21, no. 5, pp. 61-72.
- (BS84) Buchanan, B. and Shortliffe, E. (1984). Rule-Based Expert Systems: *The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, ISBN: 0201101726. Available online at: <http://www.aaai.org/Classic/Buchanan/buchanan.html>.
- (Bus00) The Business Rules Group (2000). *Defining Business Rules; What Are They Really?* (a.k.a. The GUIDE Business Rules Project, Final Report) revision 1.3. Available online at <http://www.businessrulesgroup.org>
- (Bus05) The Business Rules Team (2005). *Semantics of Business Vocabulary and Business Rules (SBVR)*. Response to RfP: *Business Semantics of Business Rules* (OMG Document br/2003-06-03). Status: finalization underway. Available online at: <http://www.omg.org>
- (Byr95) Byrd, T.A. (1995). Expert systems implementation: interviews with knowledge engineers. *Industrial Management & Data Systems*. Vol. 95, no. 10, pp. 3-7.
- (B+07) Berstel, B., Bonnard, P., Bry, F., Eckert, M. and Pătrânjan, P. L. (2007). Reactive Rules on the Web. In *Lecture Notes in Computer Science*. Vol. 4636, pp. 183-239.
- (B+06a) Bry, F., Eckert, M., Pătrânjan, P. L. and Romanenko, I. (2006). Realizing Business Processes with ECA Rules: Benefits, Challenges, Limits. In *Lecture Notes in Computer Science*. Vol. 4187, pp. 48-62.

- (B+90) Bell, J., Brooks, D., Goldbloom, E., Sarro, R. and Wood, J. (1990). *Re-Engineering Case Study Analysis of Business Rules and Recommendations for Treatment of Rules in a Relational Database Environment*, Bellevue Golden: US West Information Technologies Group.
- (Chi04) Chisholm, M. (2004). *How to Build a Business Rules Engine: Extending Application Functionality Through Metadata Engineering*. Morgan Kaufmann, ISBN: 1558609180.
- (Dat00) Date, C.J. (2000). *What Not How: The Business Rules Approach to Application Development*. Addison-Wesley, ISBN: 0201708507.
- (Deb05) Debevoise, T. *Business Process Management With a Business Rules Approach: Implementing the Service Oriented Architecture*. Business Knowledge Architects, ISBN: 0976904802.
- (Die05) Dietrich, J. and Paschke, A. (2005). On the Test-Driven Development and Validation of Business Rules. In *Proceedings of ISTA'05*, Massey, New Zealand.
- (Dur94) Durkin, J. (1994). *Expert Systems: design and development*. Macmillan Publishing Company, ISBN: 0023309709.
- (D+77) Davis, R., Buchanan, B. and Shortliffe, E. (1977). Production Rules as a Representation for a Knowledge-Based Consultation Program. *Artificial Intelligence*. Vol. 8, pp. 15-45.
- (Erl05) Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology and Design*. Prentice Hall, ISBN: 0131858580.
- (Flo97) Flood, G. (1997). Business Rules and Java still just getting acquainted. *Computer Business Review (CBR) Online*, no. 3215. URL: http://www.cbronline.com/article_cg.asp?guid=FA8B80BA-9AF8-493F-8FE8-B3ACE5E3D560
- (For82) Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In *Artificial Intelligence*. Vol. 19, pp. 17-37.
- (Gol07) Goldberg, L. (2007). *A SOA-based Business Rules Approach: Decision Services*. URL: <http://www.bpminstitute.org/articles/article/article/a-soa-based-business-rules-approach-decision-services.html>
- (Gra06) Graham, I. (2006). *Business Rules Management and Service-Oriented Architecture: a Pattern Language*. Wiley & Sons, ISBN: 0470027215.
- (Kru99) Kruchten, P. (1999). *The Rational Unified Process*. Addison-Wesley, 2nd ed. ISBN: 0201707101.
- (Hal01) Von Halle, B. (2001). *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley & Sons, ISBN: 0471412937.

- (Her95) Herbst, H. (1995). A Meta-Model for Specifying Business Rules in Systems Analysis. In Iivari, J., Lyytinen, K., Rossi, M. (eds.). *Proceedings of the Seventh Conference on Advanced Information Systems Engineering (CAiSE'95)*, Springer, pp. 186-199.
- (HG06) Von Halle, B. and Goldberg, L. (editors) (2006). *The Business Rule Revolution: Running Business the Right Way*. Happy About, ISBN: 1-60005-013-1.
- (Hoo07) Van Hooidonk, N. BiSi. Technical report (unpublished), available at Logica.
- (Hoo08) Van Hooidonk, N., Principle Consultant BPM/SOA at Logica. Personal communication, 10 March 2008.
- (ILO08) ILOG S.A. (2008). URL: <http://www.ilog.com>
- (Lae08) De Laender, F., Software Architect at Logica. Personal communication, 7 March 2008.
- (LG91) Lucas, P. and Van der Gaag, L. (1991). *Principles of Expert Systems*. Addison-Wesley, ISBN: 0201416409.
- (Min75) Minsky, M. (1975). A Framework for Representing Knowledge. In Winston (ed.): *The Psychology of Computer Vision*. McGraw-Hill, New York.
- (Mor02) Morgan, T. (2002). *Aligning IT with Business Goals*. Addison-Wesley, ISBN: 0201743914.
- (Oas06) OASIS (2006). *Reference Model for Service Oriented Architecture 1.0*. URL: <http://www.oasis-open.org>
- (PLL98) do Prado Leite, J.C.S. and Leonardi, M.C. (1998). Business Rules as Organizational Policies. In *Proceedings of the Ninth International Workshop on Software Specification and Design*, Los Alamitos: IEEE Computer Soc., pp. 68-76.
- (Pol58) Polanyi, M. (1958). *Personal Knowledge: Towards a Post-Critical Philosophy*. University of Chicago Press, ISBN: 0226672883.
- (PVT99) Pol, M., van Veenendaal, E. and Teunissen, R. (1999). *Testen volgens TMap* (2nd ed.). Tutein Nolthenius, ISBN: 9072194586.
- (RN91) Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, ISBN: 0071008942.
- (RN03) Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, 2nd edition. Prentice Hall, ISBN: 0130803022.
- (Roy70) Royce, W. (1970). Managing the Development of Large Software Systems. In *Proceedings of IEEE WESCON* no. 26, pp. 1-9.

- (Sow91) Sowa, J. F. (1991) (ed.). *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann Publishers, San Mateo, CA.
- (Ste90) Steels, L. (1990). Components of Expertise. *AI Magazine*, vol. 11, no. 2, pp. 30-49.
- (SZ92) Sowa, J. F. and Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*, vol. 31, no. 3, pp. 590-616.
- (S+99) Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W. and Wielinga, B. (1999). *Knowledge Engineering and Management: The CommonKADS methodology*. The MIT Press, ISBN: 0262193000.
- (Tur05) Turban, E., Aronson, J. and Liang, T.P. (2005). *Decision Support Systems and Intelligent Systems*, 7th edition. Prentice-Hall, ISBN: 0130461067.
- (WSB92) Wielinga, B. J., Schreiber, A. Th. and Breuker, J. A. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, vol. 4, no. 1, pp. 5-53.
- (Zac87) Zachman, J. A. (1987). *A Framework for Information Systems Architecture*. *IBM Systems Journal*, vol. 26, no. 3, pp. 276-292.

A Appendix: BKR field codes

Below a list is presented (in Dutch) of the credit codes and flags that are used by the BKR.

Possible credit types (field E) are:

- RK: doorlopend krediet met een termijnmelding na vervaldatum van 2 tot 4 maanden
- AK: aflopend krediet met een termijnmelding na vervaldatum van 2 maanden
- VK: verzendhuiskrediet met een termijnmelding na vervaldatum van 3 maanden
- HY: hypotheek ter financiering van eigen woning met een termijnmelding na vervaldatum van 120 dagen
- SR: schuldregeling met een termijnmelding na vervaldatum van 2 maanden
- RO: overige obligo's (zoals een ongeoorloofde roodstand op een betaalrekening) met een termijnmelding na vervaldatum van 4 maanden
- TC: telecommunicatievoorziening met een termijnmelding na vervaldatum van 60-120 dagen.

Possible flags (field F) are:

- A: Er is een betalingsachterstand opgetreden
- H: De achterstand is ingelopen en het krediet blijft doorlopen
- 1: Er is een aflossings- of schuldregeling getroffen nadat de achterstand / overstand situatie is ontstaan
- 2: De (restant)vordering is opeisbaar gesteld
- 3: Er is een bedrag van 250,00 EUR of meer afgeboekt. Als de afboeking wegens finale kwijting plaatsvindt, wordt tegelijkertijd met de 3 een einddatum gemeld
- 4: De kredietnemer blijkt / bleek onbereikbaar.

Source: <http://www.bkr.nl>